IMPLEMENTASI PROTOKOL 6LOWPAN PADA PURWARUPA WIRELESS SENSOR NODES

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Faisal Akhmadi
NIM: 115060900111022

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI PROTOKOL 6LOWPAN PADA PURWARUPA WIRELESS SENSOR NODES

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Faisal Akhmadi
NIM: 115060900111022

Skripsi ini telah diuji dan dinyatakan lulus pada
03 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing 1	Dosen Pembimbing 2
Sabriansyah Rizqika Akbar, S.T., M.Eng	Achmad Basuki, S.T, M.MG, Ph.D
NIP. 19820809 201212 1 004	NIP. 19741118 200312 1 002

Mengetahui
Ketua JurusanTeknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D.
NIP: 19710518 200312 1 001

IDENTITAS TIM PENGUJI

- **Penguji 1**
  Adhitya Bhawiyuga, S.Kom, M.Sc
  NIK. 201405 890720 1 001
- **Penguji 2**
  Eko Sakti Pramukantoro, S.Kom, M.Kom
  NIK. 201102 860805 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik disuatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 03 Agustus 2018

Faisal Akhmadi
115060900111022

RESUME / CURRICULUM VITAE

Biodata

Nama : Faisal Akhmadi

Alamat : Jl. Sagu No 4, Gresik

No. HP : 0897-3821-162

Email : f4khmadi@gmail.com

Pendidikan

College Summary	***Graduation Year***
Brawijaya University, Informatics Engineering	-
SMA Negeri 1 Manyar	2011
SMP Negeri 2 Gresik	2008
SD Muhammadiyah 2 Gresik	2005

UCAPAN TERIMA KASIH

Dalam penyusunan dan penulisan skripsi ini tidak terlepas dari bimbingan serta dukungan dari berbagai pihak. Oleh karena itu penulis menyampaikan banyak terima kasih kepada:

1. Kedua orang tua serta keluarga penulis yang sudah memberikan doa, dukungan, dan kepercayaannya kepada penulis.
2. Bapak Wayan Firdaus Mahmudy, S.Si, M.T., Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
3. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
4. Bapak Sabriansyah Rizqika Akbar, S.T., M.Eng. selaku Ketua Program Studi Teknik Komputer Fakultas Ilmu Komputer Universitas Brawijaya Malang, Pembimbing Akademik, dan Dosen Pembimbing I skripsi ini.
5. Bapak Achmad Basuki, S.T, M.MG, Ph.D selaku Dosen Pembimbing Akademik sekaligus Dosen Pembimbing II skripsi ini.
6. Bapak Adharul Muttaqin, S.T., M.T. selaku Dosen Pembimbing Akademik.
7. Rekan-rekan mahasiswa seperjuangan yang selalu berbagi dukungan dan motivasi selama masa perkuliahan.
8. Seluruh teman-teman lainnya yang tidak bisa penulis sebutkan satu persatu.

ABSTRAK

Sejalan dengan perkembangan teknologi *Wireless Sensor Network* (*WSN*) muncul sebagai sebuah konfigurasi jaringan yang mampu menyediakan solusi untuk beragam aplikasi dalam berbagai situasi yang sulit atau tak terduga. Jaringan ini telah banyak digunakan di berbagai sektor yang membutuhkan deteksi perubahan lingkungan fisik atau kimia. Secara umum ada dua standar komunikasi *wireless* yang digunakan dalam *WSN*, yaitu *IEEE 802.11* dan *IEEE 802.15.4*. Masing-masing standar tersebut mampu menghadirkan penggunaan *Internet Protocol* (*IP*) pada perangkat WSN. Penggunaan protokol *IP* pada *WSN* akan membawa berbagai keuntungan yang sama dengan *IP* pada jaringan komputer yang telah lama digunakan, dengan berbagai mekanisme dan protokol yang telah dikembangkan, divalidasi dan digunakan operasional. Dengan tidak adanya blok alamat *IPv4* yang tersedia, penggunaan alamat *IPv6* harus segera dimulai. Tetapi, untuk membawa *IPv6* ke *WSN* akan terhambat dengan terbatasnya sumber daya. Standar IEEE 802.15.4 memungkinkan perangkat dengan daya rendah dan koneksi yang terbatas *(lossy network)* untuk terhubung dengan *IP network.* Konsep *IPv6 over Low-power Wireless Personal Area Network (6LoWPAN)* hadir untuk memungkinkan implementasi protokol *internet* ke perangkat yang kecil sekalipun. Dalam penelitian ini dilakukan perancangan dan implementasi purwarupa *wireless sensor* menggunakan MRF24J40MA sebagai modul *radio transceiver* yang menggunakan 6LoWPAN sebagai protokol komunikasinya. Ada fitur yang disematkan dalam sistem ini yang memungkinkan *sensor node* mengetahui alamat *IP server* tanpa konfigurasi manual. Perangkat keras yang digunakan dalam penelitian ini adalah Raspberry Pi dan MRF24J40MA. Akan dibahas mengenai bagaimana pemasangan MRF24J40MA hingga bisa digunakan untuk media komunikasi menggunakan protokol *6LoWPAN.* Hasil pengujian dari penelitian ini menunjukkan bahwa secara keseluruhan, sistem yang telah dibuat dapat berjalan sesuai fungsional dan bisa menggunakan *6LoWPAN* sebagai protokol komunikasi.

Kata kunci: *6LoWPAN, WSN, multicast,* Raspberry Pi, MRF24J40MA

ABSTRACT

In line with the development of technology Wireless Sensor Network (WSN) is used as a network that is able to provide solutions for various applications in various unexpected difficulties. This network has been widely used in various sectors that require physical or chemical environments. In general, there are two wireless communication standards in WSN, namely IEEE 802.11 and IEEE 802.15.4. Each standard uses Internet Protocol (IP) on WSN devices. The use of the IP protocol at WSN will produce various benefits similar to IP on a long computer network, with various kinds and that have been developed, validated and operational. In the absence of blocks of available IPv4 addresses, the use of IPv6 addresses must begin immediately. However, to bring IPv6 to WSN will be hampered by limited resources. The IEEE 802.15.4 standard allows devices with low power and limited connections to connect to an IP network. The concept of IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) is here to enable the implementation of internet protocols to even small devices. In this study, the design and implementation of a wireless sensor prototype using MRF24J40MA as a transceiver radio module that uses 6LoWPAN as the communication protocol is carried out. There are features added in this system that allow the sensor node to know the server's IP address without manual configuration. The hardware used in this study is Raspberry Pi and MRF24J40MA. Also discussed about how to install MRF24J40MA so that it can be used for communication media using the 6LoWPAN protocol. The test results from this study indicate that overall, the system that has been made can run functionally and can use 6LoWPAN as a communication protocol.

Keywords: *6LoWPAN, WSN, multicast, Raspberry Pi*, *MRF24J40MA*

KATA PENGANTAR

Puji syukur kehadirat Allah SWT, berkat rahmat dan karunia-Nya penulis dapat menyelesaikan penyusunanskripsi dengan baik. Shalawat serta salam semoga senantiasa terlimpahkan kepada Nabi Muhammad SAW. Penulisan skripsi ini diajukan untuk memenuhi salah satu syarat untuk mendapatkan gelar Sarjana pada Fakultas Ilmu Komputer, Universitas Brawijaya Malang. Judul skripsi yang penulis ajukan adalah "Implementasi 6LoWPAN Pada Purwarupa Wireless Sensor Node".

Dalam penyusunan dan penulisan skripsi ini tidak terlepas dari bimbingan serta dukungan dari berbagai pihak. Oleh karena itu penulis menyampaikan banyak terimakasih kepada yang terhormat:

9. Kedua orang tua serta keluarga penulis yang sudah memberikan doa, dukungan, dan kepercayaannya kepada penulis.
10. Bapak Wayan Firdaus Mahmudy, S.Si, M.T., Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
11. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
12. Bapak Sabriansyah Rizqika Akbar, S.T., M.Eng. selaku Ketua Program Studi Teknik Komputer Fakultas Ilmu Komputer Universitas Brawijaya Malang, Pembimbing Akademik, dan Dosen Pembimbing I skripsi ini.
13. Bapak Achmad Basuki, S.T, M.MG, Ph.D selaku Dosen Pembimbing Akademik sekaligus Dosen Pembimbing II skripsi ini.
14. Bapak Adharul Muttaqin, S.T., M.T. selaku Dosen Pembimbing Akademik.
15. Rekan-rekan mahasiswa seperjuangan yang selalu berbagi dukungan dan motivasi selama masa perkuliahan.
16. Seluruh teman-teman lainnya yang tidak bisa penulis sebutkan satu persatu.

Malang, 03 Agustus 2018

Faisal Akhmadi
115060900111022@mail.ub.ac.id

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Sejalan dengan perkembangan teknologi *Wireless Sensor Network* (*WSN*) muncul sebagai sebuah konfigurasi jaringan yang mampu menyediakan solusi untuk beragam aplikasi dalam berbagai situasi yang sulit atau tak terduga. Jaringan ini telah banyak digunakan di berbagai sektor yang membutuhkan deteksi perubahan lingkungan fisik atau kimia (Yick, et al., 2008). Ada beberapa tantangan yang harus diatasi dalam *WSN*, salah satunya adalah mengenai sumber dayanya yang terbatas. Secara umum perangkat *WSN* mempunyai ukuran fisik kecil dan bergantung dengan kapasitas daya baterai yang mengakibatkan terbatasnya sumberdaya. Ketika *node WSN* disebarkan untuk berbagai aplikasi, konfigurasi secara otomatis adalah sebuah keharusan, karena ketika ditempatkan dalam lingkungan yang sulit dijangkau akan mengakibatkan sulitnya untuk mengontrol jaringan atau perangkat. Masalah yang biasanya muncul adalah ketika digunakan model *client-server*, dimana secara umum alamat *server* diberikan kepada *client* sebelum perangkat disebarkan. Hal ini akan menjadi masalah ketika alamat *server* berubah dan *client* tidak bisa dijangkau, baik secara lokasi atau lainnya.

Secara umum ada dua standar komunikasi *wireless* yang digunakan dalam *WSN*, yaitu *IEEE 802.11* dan *IEEE 802.15.4*. Masing-masing standar tersebut mampu menghadirkan penggunaan *Internet Protocol* (*IP*) pada perangkat WSN. Penggunaan protokol *IP* pada *WSN* akan membawa berbagai keuntungan yang sama dengan *IP* pada jaringan komputer yang telah lama digunakan, dengan berbagai mekanisme dan protokol yang telah dikembangkan, divalidasi dan digunakan operasional (Mayer & Fritsche, 2006). Dengan tidak adanya blok alamat *IPv4* yang tersedia (Babatunde & Al-Debagy, 2014), penggunaan alamat *IPv6* harus segera dimulai. Tetapi, untuk membawa *IPv6* ke *WSN* akan terhambat dengan terbatasnya sumber daya. Standar IEEE 802.15.4 memungkinkan perangkat dengan daya rendah dan koneksi yang terbatas *(lossy network)* untuk terhubung dengan *IP network* (Ott, 2012). Konsep *IPv6 over Low-power Wireless Personal Area Network (6LoWPAN)* hadir untuk memungkinkan implementasi protokol *internet* ke perangkat yang kecil sekalipun (Mulligan, 2007). Meskipun sama-sama berdasarkan *IEEE 802.15.4, 6LoWPAN* tidak sepopuler jaringan lainnya, seperti *Zigbee* misalnya. Tetapi, *6LoWPAN* menggunakan *IPv6,* dan ini saja seharusnya bisa memberikan keuntungan yang lebih.

Berdasarkan latar belakang tersebut penelitian ini akan ditunjukkan untuk mengimplementasikan protokol *6LoWPAN* dengan MRF24J40MA sebagai modul *radio transceiver* pada purwarupa *wireless sensor nodes* dengan penambahan fitur yang memungkinkan *node* mengetahui alamat *IP server* tanpa perlu konfigurasi secara manual.

1.2 Rumusan Masalah

1. Bagaimana cara merancang purwarupa *wireless sensor nodes* yang menggunakan *6LoWPAN* sebagai protokol komunikasinya?
2. Bagaimana cara agar *node* bisa mengetahui alamat *IP server* tanpa perlu konfigurasi secara manual?
3. Bagaimana uji fungsionalitas dari purwarupa *wireless sensor nodes* yang menggunakan protokol *6LoWPAN*?

1.3 Tujuan

Tujuan dari penelitian ini adalah sebagai berikut:

1. Merancang purwarupa wireless sensor nodes yang menggunakan 6LoWPAN sebagai protokol komunikasinya.
2. *Node* bisa mengetahui alamat *IP server* tanpa perlu konfigurasi secara manual.
3. Mengetahui fungsionalitas dari purwarupa *wireless sensor nodes* yang telah diimplementasikan.

1.4 Manfaat

Manfaat dari penelitian ini, bisa digunakan sebagai contoh implementasi dari protokol *6LoWPAN* pada sistem *wireless sensor nodes* menggunakan MRF24J40MA sebagai modul *radio transceiver.* Pengaplikasian protokol *6LoWPAN* akan membawa berbagai manfaat pada sistem, salah satunya adalah penggunaan *IPv6* sebagai pengalamatan jaringan. Fitur yang ditambahkan pada sistem ini akan memungkinkan *node* mengetahui alamat pengiriman data tanpa perlu dilakukan konfigurasi secara manual.

1.5 Batasan Masalah

Untuk memfokuskan permasalahan yang telah dirumuskan, maka penelitian ini dibatasi dalam hal:

1. Penelitian ini menitikberatkan pada implementasi di *layer application.*
2. Penelitian ini mengabaikan aspek keamanan pada jaringan.
3. Penelitian ini mengabaikan implementasi *routing protocol* pada *layer network*.

1.6 Sistematika Pembahasan

Sistematika penulisan penelitian ditunjukkan untuk memberikan gambaran dan uraian dari penyusunan tugas akhir secara garis besar yang meliputi beberapa bab, sebagai berikut:

BAB I : Pendahuluan

Bab ini merupakan dasar dari penyusunan skripsi yang terdiri dari latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika pembahasan.

BAB II : Kajian Pustaka dan Dasar Teori

Bab ini mambahas tinjauan pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi "*Implementasi Protokol 6LoWPAN Pada Purwarupa Wireless Sensor Nodes*".

BAB III : Metodologi

Bab ini menguraikan tentang metode dan langkah kerja yang terdiri dari studi literatur, analisis kebutuhan simulasi, perancangan sistem, implementasi dan analisis serta pengambilan kesimpulan.

BAB IV : Rekayasa Kebutuhan Sistem

Bab ini menguraikan tentang segala kebutuhan dan melakukan gambaran tentang segala aktifitas yang dilakukan untuk merancang sistem yang dibuat.

BAB V : Perancangan dan Implementasi

Bab ini menguraikan proses implementasi dari dasar teori yang telah dipelajari sesuai analisis dan perancangan sistem.

BAB VI : Pengujian dan Analisis

Bab ini memuat hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

BAB VII : Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian program, serta saran-saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi sumber pustaka yang membahas penelitian yang ada dan diusulkan. Serta dasar teori yang berisi uraian dan pembahasan tentang konsep, model, metode, atau sistem dari literatur ilmiah yang digunakan pada penelitian ini.

2.1 Penelitian Terkait

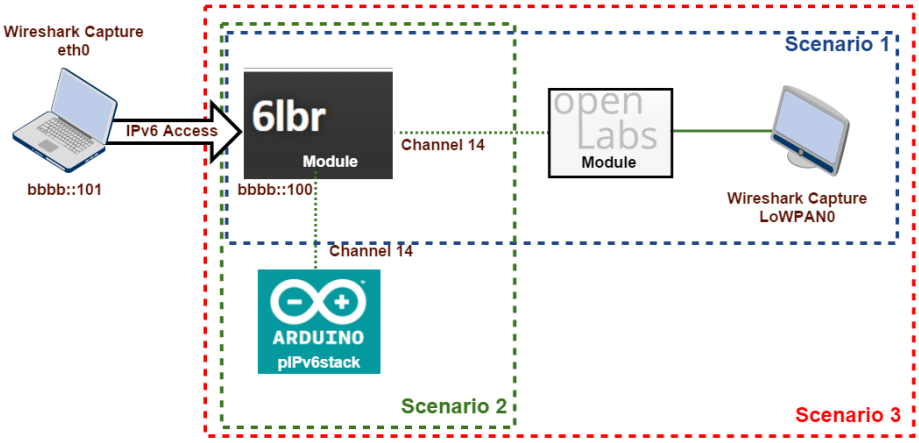
Sumber pustaka membahas penelitian yang telah ada dan yang diusulkan. Pada penelitian ini kajian pustaka diambil dari beberapa penelitian yang pernah dilakukan untuk digunakan sebagai acuan dalam penyusunan kerangka penelitian.

2.1.1 Comparative assessments of *IEEE 802.15.4*/*ZigBee* and *6LoWPAN* for low-power industrial *WSNs* in realistic scenarios oleh Emanuele Toscano dkk.

Penelitian yag dilakukan Emanuele Toscano dkk, dilakukan penilaian kinerja komparatif antara *ZigBee* dan *6LoWPAN*. Penilaian yang dilakukan adalah *end-to-end delay,* waktu *update,* dan rasio pengiriman dengan dua skenario. Pada skenario pertama dilakukan dengan periode pengiriman data 20 detik, sedangkan pada skenario kedua dilakukan dengan 400 milidetik. Hasil dari penelitian ini menunjukkan bahwa jaringan *ZigBee* mapu mendukung siklus tugas yang lebih kecil, memberikan *end-to-end delay* yang paling sedikit, dan waktu *update* yang sedikit lebih dekat dengan nilai teoritis. Sedangkan jaringan *6LoWPAN* menunjukkan nilai rata-rata *end-to-end delay* yang lebih kecil dan memiliki kehandalan yang lebih tinggi, seperti contohnya dalam hal hilangnya paket data. Hasil penelitian ini menjadi basis penulis dalam pemilihan *6LoWPAN* dengan pertimbangan kehandalan dan kebutuhan *IPv6.*

2.1.2 Internet of things implementation with Raspberry Pi oleh Alex Nasarre Ramírez

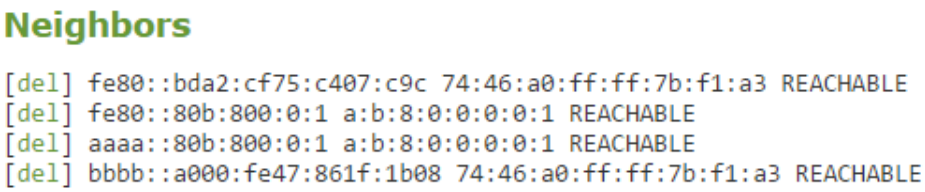
Penelitian yang dilakukan oleh Alex Nasarre Ramírez membandingkan berbagai solusi *6LoWPAN* untuk diimplementasikan ke Raspberry *Pi*. Beberapa solusi *6LoWPAN* yang dibahas adalah cetic-6LBR dan *Arduino* pIPv6 stack.



Gambar 2.1 Skenario pengujian 6LBR

Sumber: Nasarre Ramírez, 2014

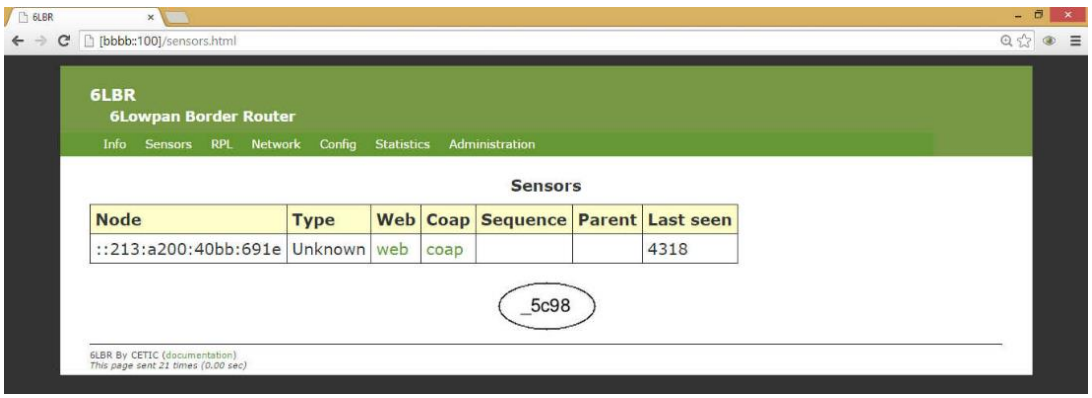
Dilakukan beberapa skenario pengujian seperti pada Gambar 2.1, pada percobaan pertama digunakan Raspberry *Pi* dan *modul radio* OpenLabs sebagai *slip-radio* untuk implementasi 6LBR. Proses instalasi berjalan dengan lancar namun terdapat kendala pada *modul radio* OpenLabs yang meskipun terdeteksi oleh Wireshark dengan keadaan *reachable* seperti pada Gambar 2.2 tetapi sebaliknya ketika dilakukan *ping*.



Gambar 2.2 *Reachable neighbors* yang terdeteksi oleh *webserver* 6LBR

Sumber: Nasarre Ramírez, 2014

Pada skenario kedua, digunakan Arduino Due, XBee *shield,* dan XBee menggunakan Arduino pIPv6 stack. Proses instalasi berjalan lancar dan terdeteksi oleh 6LBR *webserver* seperti yang ditunjukkan pada Gambar 2.3.



Gambar 2.3 *Sensor* (XBee) terdeteksi oleh 6LBR *webserver*

Sumber: Nasarre Ramírez, 2014

Hasil dari percobaan skenario dua ini berbeda dengan yang pertama, dimana *modul* OpenLabs terdeteksi dengan keadaan *stale,* modul XBee terdeteksi dengan keadaan *reachable* seperti yang ditunjukkan pada Gambar 2.4. Hasil dari penelitian ini adalah mempertimbangkan penggunaan 6LBR, Arduino pIPv6 stack sebagai yang terbaik untuk digunakan. Hasil dari penelitian tersebut dijadikan penulis sebagai dasar untuk implementasi *6LoWPAN* menggunakan Raspberry Pi.

Gambar 2.4 Status *sensor* yang terdeteksi oleh 6LBR

Sumber: Nasarre Ramírez, 2014

2.2 Dasar Teori

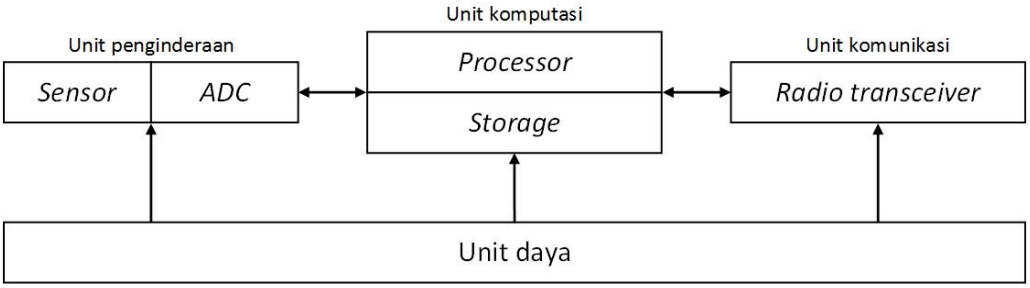
Dasar teori berisi penjelasan mengenai bahan, konsep dan objek yang dibutuhkan dalam penyusunan penelitian ini.

2.2.1 *Wireless Sensor Network* (*WSN*)

Wireless sensor network secara umum dapat digambarkan sebagai jaringan yang dibentuk oleh *node* yang secara kooperatif dapat mendeteksi perubahan fisik pada lingkungan dan memungkinkan interaksi antara manusia atau komputer dengan lingkungan sekitarnya (Bröring, et al., 2011). *WSN* memiliki karakteristik tersendiri dalam sistemnya, yang dapat diidentifikasikan sebagai berikut (Zheng & Jamalipour, 2009):

1. Penyebaran *sensor node* dengan skala besar: *sensor node* biasanya di disebarkan dengan jumlah yang banyak sesuai dengan penggunaannya.
2. Hemat energi: energi yang digunakan dalam *WSN* ditujukan untuk berbagai hal, seperti komputasi, komunikasi, dan penyimpanan. Sebuah *sensor node* mengkonsumsi lebih banyak energi untuk tujuan komunikasi dibanding tujuan lainnya. Jika sebuah *node* kehabisan energi, akan membuatnya tidak valid, karena seringkali tidak adanya pilihan untuk mengisi ulang baterai. Pengembangan protokol dan algoritma harus mempertimbahan konsumsi daya pada *node*.

3. Kemampuan daya, komputasi, dan penyimpanan yang terbatas: kemampuan komputasi, daya, dan penyimpanan pada *sensor node* biasanya terbatas. Hal ini dikarenakan pertimbangan biaya yang harus dikeluarkan, kebutuhan energi yang diperlukan, dan ukuran fisik dari *sensor node*.
4. Kemampuan mengorganisir: ketika *sensor node* dikerahkan secara acak, mereka harus bisa mengorganisir dirinya sendiri tanpa ada pengawasan. *Sensor node* bisa berkolaborasi untuk menyesuaikan diri dengan algoritma terdistribusi dan membentuk jaringan secara otomatis.
5. Berorientasi aplikasi: yang membedakan jaringan *wireless sensor* dengan jaringan konvensional lainnya adalah tujuan pemakaiannya. Mulai dari kebutuhan militer hingga kesehatan. *Sensor node* dikerahkan sesuai dengan kebutuhan tersebut.
6. *Sensor node* yang tidak bisa diandalkan: pada umumnya, *sensor node* disebarkan di lingkungan yang tidak bersahabat dan tidak dilakukan pengawasan lebih lanjut.
7. Topologi jaringan yang dinamis: *sensor node* bisa mengalami keadaan tidak valid dikarenakan kehabisan baterai atau kesalahan lainnya, saluran komunikasi bisa terganggu, dan sebuah *node* baru bisa ditambahkan dalam jaringan yang mengakibatkan perubahan pada topologi jaringan. Hal ini mengakibatkan keharusan sistem untuk bisa melakukan konfigurasi ulang dan penyesuaian secara otomatis.
8. Tidak ada identifikasi *node* secara global: karena jumlahnya yang banyak, untuk membuat skema pengalamatan global akan cukup sulit, karena akan menyebabkan permasalahan *overhead* ketika dilakukan kegiatan pemeliharaan mengenai identifikasi.

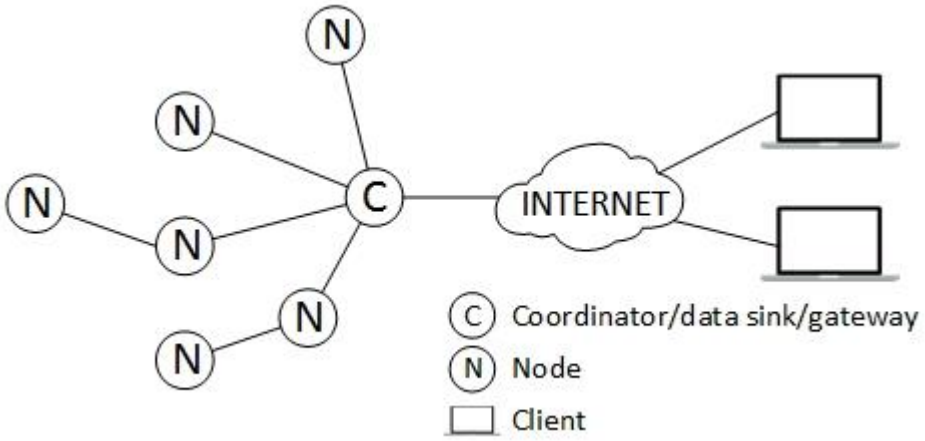


Gambar 2.5 Diagram blok arsitektur perangkat keras *sensor node*

Sumber: Ahmed, et al., 2012

WSN mempunyai lingkungan yang heterogen, baik dari segi perangkat keras maupun lunak. Perangkat keras *WSN* dapat terdiri dari berbagai jenis *sensor node* yang berbeda. Perangkat keras tersebut tersusun dari beberapa bagian, seperti yang ditunjukkan pada Gambar 2.5, yang akan dijabarkan sebagai berikut (Ahmed, et al., 2012):

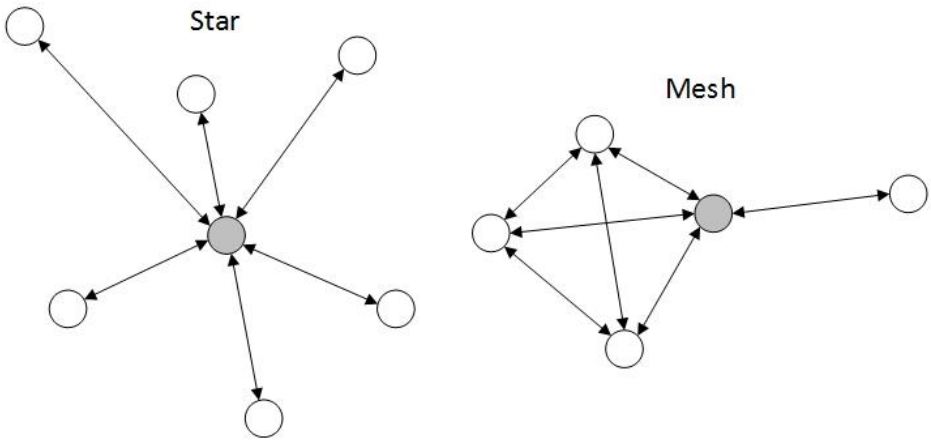
1. Unit daya: peran dari unit daya adalah menyediakan energi yang dibutuhkan oleh *sensor node*. Unit daya dibutuhkan untuk mengatur efisiensi penggunaan baterai (jika menggunakan).
2. Unit komputasi: unit ini terdiri dari *processor* (*microcontroller*) dan *storage* (*RAM*). Mempunyai peran untuk mengumpulkan dan mengolah data.
3. Unit penginderaan: unit ini terdiri dari satu atau lebih perangkat penginderaan (*sensor*) yang diperlukan untuk mendeteksi adanya perubahan dalam lingkungan secara fisik atau kimiawi. Unit yang digunakan disesuaikan dengan kebutuhan penggunaannya. *Analog-to-digital converter* (*ADC*) dibutuhkan untuk berkomunikasi dengan *microcontroller*, karena keluaran dari *sensor* adalah siyal analog yang harus diubah ke sinyal digital.
4. Unit komunikasi: unit ini menggunakan *radio transceiver* yang terdiri dari pemancar dan penerima sinyal. Komunikasi dilakukan melalui sebuah saluran komunikasi yang menggunakan protokol jaringan. Metode komunikasi yang digunakan disesuaikan dengan kebutuhan dari penggunaan sistem.



Gambar 2.6 Arsitektur *wireless sensor network*

Sebuah jaringan *wireless sensor* biasanya terdiri dari sejumlah besar *sensor node* yang disebarkan di sebuah daerah sesuai tujuan penggunaannya, dan ada satu atau lebih *data sink* yang terletak di dekat daerah *sensor node,* seperti yang ditunjukkan pada Gambar 2.6. *Data sink* akan mengumpulkan data dari *sensor node*, dan terkadang juga berfungsi sebagai *gateway* ke jaringan lainnya, seperti misalnya *internet.* Dari penjelasan tersebut bisa diidentifikasikan arsitektur *WSN* terdiri dari:

1. *Sensor node*
2. *Data sink, coordinator,* dan atau *gateway*



Gambar 2.7 Topologi jaringan yang digunakan pada *wireless sensor*

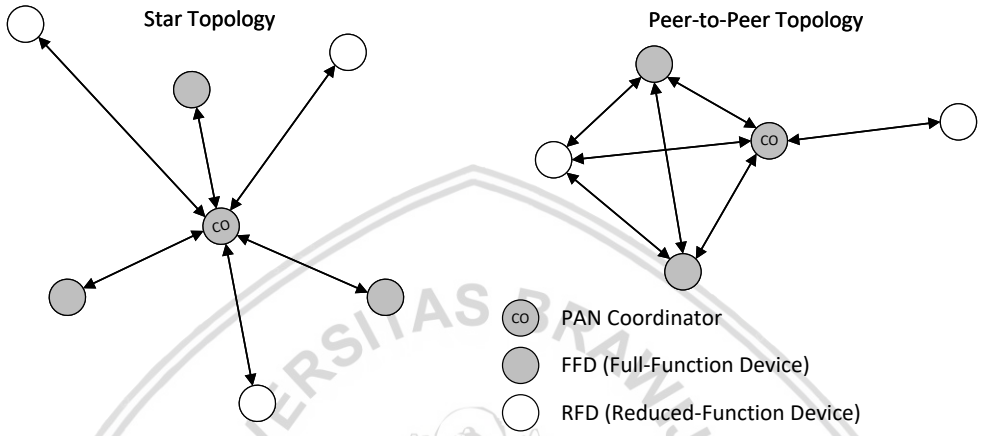
2.2.2 *IEEE 802.15.4 LR-WPAN (Low-Rate Wireless Personal Area Network)*

LR-WPAN adalah jaringan komunikasi simpel dan tidak membutuhkan biaya tinggi yang memungkinkan pengaplikasian koneksi *wireless* dengan kebutuhan daya dan *throughput* yang kecil (IEEE Computer Society, 2011). Tujuan utama dari *LR-WPAN* adalah kemudahan pemasangan, transfer data yang reliabel, kebutuhan biaya yang murah, dan masa hidup baterai yang proporsional, sembari dengan mempertahankan protokol yang sederhana dan fleksibel (IEEE Computer Society, 2011). Sebuah sistem yang sesuai dengan standar *IEEE 802.15.4* terdiri dari beberapa komponen. Yang paling dasar adalah perangkatnya. Perangkat tersebut bisa berupa *FFD* (*Full-Function* Device) atau *RFD* (*Reduced-Function Device*). Dua perangkat atau lebih dalam *POS* (*Personal Operating Space*) dan beroperasi pada *physical RF channel* yang sama, sudah bisa dikatakan sebagai *WPAN* (IEEE Computer Society, 2011). Beberapa kemampuan yang disediakan oleh standar ini adalah sebagai berikut (IEEE Computer Society, 2011):

- Topologi jaringan *star* atau *peer-to-peer.*
- *Unique 64-bit extended address* atau *allocated 16-bit short address.*
- Pilihan alokasi *guaranteed time slots* (*GTSs*).
- *Carrier sense multiple access with collision avoidance* (*CSMA-CA*) atau akses *ALOHA channel*.
- Protokol yang telah diakui sepenuhnya untuk kehandalan transfer data.
- Konsumsi daya yang rendah.
- *Energy detection* (*ED*).
- *Link quality indication* (*LQI*).

Tergantung pada kebutuhan aplikasinya, *LR-WPAN* beroperasi menggunakan topologi *star* atau *peer-to-peer,* dengan diagram seperti ditunjukkan pada Gambar 2.8. Pada topologi *star*, komunikasi dibangun antara perangkat dengan

satu perangkat kontrol yang disebut *PAN coordinator.* Semua perangkat yang terhubung pada salah satu topologi tersebut memiliki pengalamatan unik yang disebut dengan *extended address.* Sebuah perangkat bisa menggunakan *extended address* untuk berkomunikasi secara langsung dalam area *PAN* atau menggunakan *short address* yang diberikan oleh *PAN coordinator* ketika terhubung. Pada umumnya perangkat dengan fungsi sebagai *PAN coordinator* menggunakan tenaga listrik secara langsung, sementara perangkat *node* menggunakan baterai.

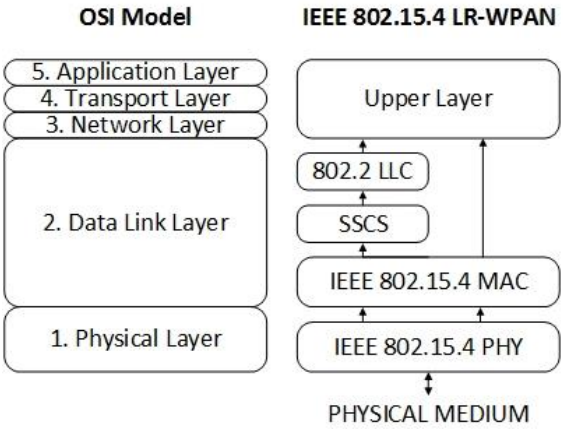


Gambar 2.8 Topologi jaringan yang digunakan pada *802.15.4 LR-WPAN*

Sumber: Olsson, 2014

Topologi *peer-to-peer* juga mempunyai *PAN coordinator,* tetapi berbeda dengan topologi *star,* dimana pada topologi *peer-to-peer* komunikasi antar perangkat (*node*) memungkinkan selama masih pada jarak jangkauan satu sama lain. Topologi *peer-to-peer* memungkinkan implementasi jaringan yang kompleks, seperti contohnya jaringan *mesh.* Topologi *peer-to-peer* memungkinkan beberapa *hop* untuk pengiriman data dari perangkat manapun menuju perangkat lain pada satu jaringan. Fungsi tersebut dapat ditambahkan pada *layer* yang lebih tinggi, karena bukan dari bagian dari standar *802.15.4 LR-WPAN.*

Setiap *PAN* mempunyai sebuah identifikasi yang unik untuk membedakannya dengan *PAN* lain yang mempunyai jarak atau area yang berdekatan, yang disebut *PAN ID.* Dengan adanya *PAN ID* ini, memungkinkan perangkat dalam satu area *PAN* berkomunikasi satu sama lain menggunakan *short address* dan pengiriman data antar *PAN.* Perangkat *LR-WPAN* terdiri dari setidaknya satu *PHY,* yang berupa *RF transceiver* beserta mekanisme kontrol level bawahnya dan *MAC sublayer* yang memberikan kemampuan akses pada *physical channel* untuk semua tipe pengiriman data. Gambar 2.9 menunjukkan gambar representasi dari blok *layer* pada standar *802.15.4 LR-WPAN.*



Gambar 2.9 *IEEE 802.15.4 LR-WPAN protocol stack*

Sumber: Olsson, 2014

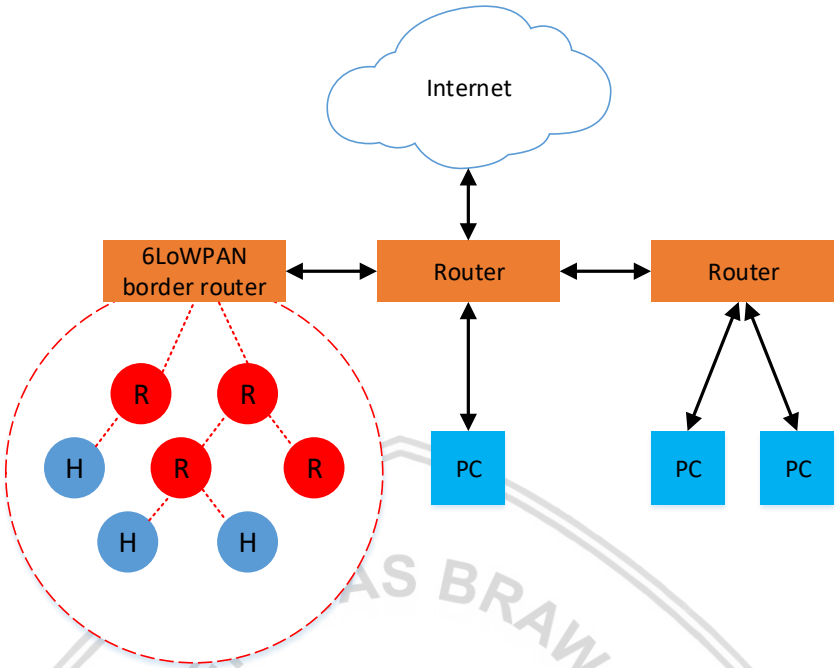
Upper layer, yang ditunjukkan pada Gambar 2.9, terdiri dari *network layer,* yang menyediakan konfigurasi jaringan dan rute pengiriman, *transport layer* yang umunya berisi protokol *UDP* dan *TCP*, dan *application layer* yang menyediakan fungsi aplikasi sesuai dengan tujuan penggunaannya. Definisi dari *upper layer* ini diluar lingkup dari standar *802.15.4 LR-WPAN* (IEEE Computer Society, 2011). Standar protokol *802.15.4* menjadi dasar dari beberapa protokol, seperti *ZigBee*, *MiWi Mesh* dan *MiWi P2*, *6LoWPAN*, *WirelessHART*, dan *ISA100.11a* (Ott, 2012).

PHY layer menyediakan dua layanan, layanan data, dan layanan pengelolaan. Layanan data *PHY* memungkinkan pengiriman dan penerimaan dari *PHY protocol data units* (*PPDU*) di saluran radio (*radio channel*). Fitur dari *PHY* adalah aktivasi dan deaktivasi *radio transceiver, energy detection, link quality indication,* pemilihan saluran (*channel*), *clear channel assessment* (*CCA*), dan pengiriman serta penerimaan paket di media fisik (IEEE Computer Society, 2011).

MAC layer juga menyediakan dua layanan, layanan data, dan layanan pengelolaan yang menyediakan antarmuka ke *MAC sublayer management entity* (*MLME*) service access point (*SAP*) (dikenal sebagai *MLME-SAP*). Layanan data pada *MAC* memungkinkan pengiriman dan penerimaan *MAC protocol data unit* (*MPDU*) di layanan data *PHY.* Fitur dari *MAC sublayer* adalah pengelolaan *beacon*, akses saluran (*channel*), pengelolaan *guaranteed time slot* (*GTS*), validasi *frame, acknowledged frame delivery, association,* dan *disassociation* (IEEE Computer Society, 2011).

2.2.3 *IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN)*

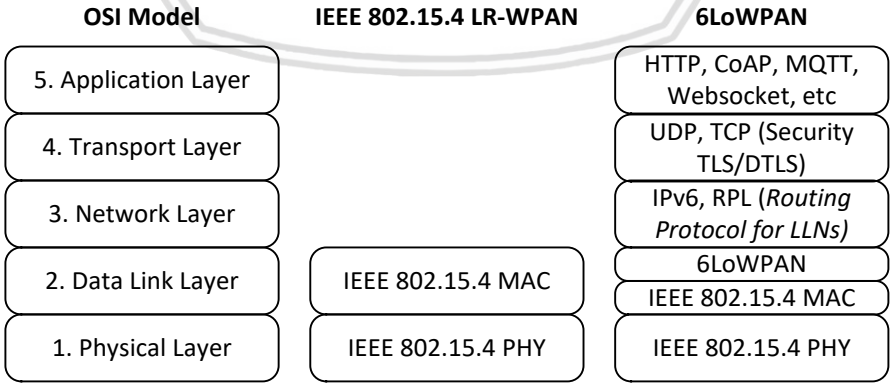
Didefinisikan oleh *Internet Engineering Task Force* (*IETF*), *6LoWPAN* adalah teknologi jaringan atau penyesuaian *layer* yang memungkinkan paket *IPv6* dibawa secara efisien dalam *link layer* dengan frame kecil (Olsson, 2014).



Gambar 2.10 Contoh jaringan mesh *6LoWPAN*

Sumber: Olsson, 2014

Gambar 2.10 menunjukkan contoh dari jaringan *mesh 6LoWPAN*. *Uplink* menuju *internet* ditangani oleh *access point (AP)* yang bertindak sebagai *rout*er *IPv6*. Jaringan *6LoWPAN* terhubung ke jaringan *IPv6* menggunakan *border router. Border router* mempunyai tiga fungsi: 1) pertukaran data antar perangkat *6LoWPAN* dan *internet*; 2) pertukaran data lokal antar perangkat *6LoWPAN*; 3) pembuatan dan pemeliharaan *subnet* jaringan *6LoWPAN*. Mengenai perbedaan dari *IEEE 802.15.4 LR-WPAN* dengan *6LoWPAN*, seperti yang ditunjukkan pada Gambar 2.11.



Sumber: Olsson, 2014

Gambar 2.11 Perbandingan *protocol stack antara IEEE 802.15.4* dengan *6LoWPAN*

Beberapa keunggulan *6LoWPAN* adalah sebagai berikut:

1. Standar terbuka yang termasuk *TCP, UDP, HTTP, CoAP, MQTT,* dan *websockets* (Olsson, 2014).
2. *End-to-end IP addressable nodes,* penggunaan *end-to-end*, infrastruktur berbasis *IP* mendapat manfaat penuh dari 30 tahun pengembangan teknologi *IP* dan memfasilitasi standar terbuka dan interoperabilitas (Olsson, 2014).
3. Tidak diperlukan *gateway, router* secara langsung menghubungkan antara *6LoWPAN* ke *IP* (Olsson, 2014).
4. Dukungan untuk topologi jaringan *mesh* yang besar, komunikasi yang kuat dan komsumsi daya yang sangat rendah (Olsson, 2014).
5. Perangkat yang terhubung ke *6LoWPAN* bisa dalam keadaan *"sleep"* dalam jangka waktu yang lama untuk menghemat energi (Kushalnagar, et al., 2007).

2.2.4 Linux-WPAN

Linux-wpan ini adalah proyek implementasi dari standar *IEEE 802.15.4* dan *6LoWPAN* yang telah ditetapkan oleh *IEEE 802.15.4 working group.* Proyek ini bermula dari proyek *linux-zigbee* yang dimulai pada tahun 2008 di *website* SourceForge. Saat ini proyek *linux-wpan* dikelola oleh Alexander Aring dari Pengutronix (Schmidt, 2015). Tujuan dari proyek ini adalah untuk menyediakan implementasi lengkap dari protokol standar *IEEE 802.15.4* dan *6LoWPAN.*

Tabel 2.1 Perangkat modul *transceiver* yang didukung oleh *linux-wpan*

Nama modul *transceiver*	Nama *driver*	Pembuat
adf7242	*adf7242*	*Analog Devices*
at86rf212	*at86rf230*	*Atmel*
at86rf212b	*at86rf230*	*Atmel*
at86rf231	*at86rf230*	*Atmel*
at86rf233	*at86rf230*	*Atmel*
atusb	*atusb*	*Atmel*
cc2520	*cc2520*	*Texas Instrument*
mrf24j40	*mrf24j40*	*Microchip*

Tidak semua perangkat keras *transceiver* bisa digunakan dengan *linux-wpan* seperti yang ditunjukkan pada Tabel 2.1, meskipun perangkat tersebut menggunakan modul radio *802.15.4*. Dalam proyek ini terdapat sebuat *tool* yang bernama *wpan-tools*. Fungsi dari *wpan-tools* ini adalah menyediakan akses ke antarmuka *netlink* perangkat keras *802.15.4*. *Tool* ini menyediakan control *userspace* untuk *user*, sehingga *user* bisa melakukan konfigurasi perangkat

802.15.4 yang digunakan. Perintah konfigurasi yang tersedia adalah *iwpan* dan *wpan-ping*. Perintah *iwpan* digunakan untuk konfigurasi modul *transceiver*, seperti memberikan alamat, *pan id*, dan sebagainya. Berikut ini adalah daftar perintah dari *iwpan:*

1. phy <phyname> set channel <page> <channel>

   Perintah ini digunakan untuk memberikan konfigurasi *channel* yang digunakan. Contohnya iwpan phy phy0 set channel 0 11, dimana phy0 adalah perangkat fisik *transceiver* dan 11 adalah *channel* yang digunakan.

2. phy <phyname> set tx_power <dBm>

   Perintah ini digunakan untuk memberikan konfigurasi kekuatan sinyal yang digunakan oleh *transceiver*, dalam bentuk *dBm.*

3. dev <devname> set pan_id <pan_id>

   Perintah ini digunakan untuk memberikan *pan id*. Contohnya sudo iwpan dev wpan0 set pan_id 0x24, dimana wpan0 adalah nama *interface* dan 0x24 adalah *pan id.*

4. dev <devname> set short_addr <short_addr>

   Perintah ini digunakan untuk memberikan *short address* pada *interface.*

5. phy <phyname> info

   Perintah ini digunakan untuk menampilkan kemampuan atau spesifikasi dari perangkat fisik *transceiver*.

6. list

   Perintah ini digunakan untuk menampilkan daftar semua perangkat fisik *transceiver* yang terpasang beserta kemampuan atau spesifikasinya.

Daftar perintah ini hanyalah sebatas yang penulis butuhkan atau gunakan ketika melakukan penelitian. Perintah *iwpan* pada *linux-wpan* ini membagi perangkat yang dapat dikonfigurasi ke dalam *phy* (perangkat fisik) dan perangkat *wpan* (*network interfaces*). Bisa dilihat pada daftar perintah sebelumnya, perintah *phy* hanya digunakan untuk konfigurasi kemampuan atau fitur perangkat keras. Sedangkan perintah *dev* digunakan untuk memberikan konfigurasi *network interface*, seperti memberi alamat, channel, dan sebagainya.

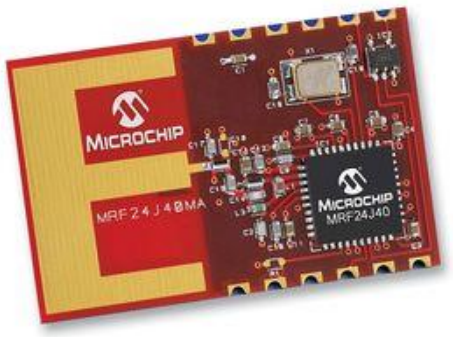
2.2.5 *Router Advertisement Daemon* (RADVD)

Radvd adalah sebuah *daemon* bertindak sebagai *router IPv6* yang berjalan pada sistem Linux atau BSD. Pesan *Router Advertisement* akan dikirimkan secara berkala ke jaringan local atau ke *node* ketika *node* tersebut mengirim pesan *Router Solicitation.* Proses *Router Discovery* ini menghasilkan *stateless address auto-configuration,* dimana tidak diperlukan konfigurasi secara manual pada sisi *client* atau *nodes.* Untuk menggunakan Radvd hanya dibutuhkan instalasi dan konfigurasi seperti pada Tabel 2.2.

Tabel 2.2 Contoh konfigurasi Radvd

```
1  interface eth0 {
2                 AdvSendAdvert on;
3                 MinRtrAdvInterval 3;
4                 MaxRtrAdvInterval 10;
5                 prefix 3ffe:ffff:0100:f101::/64 {
6                                 AdvOnLink on;
7                                 AdvAutonomous on;
8                                 AdvRouterAddr on;
9                  };
10 };
```

2.2.6 MRF24J40MA

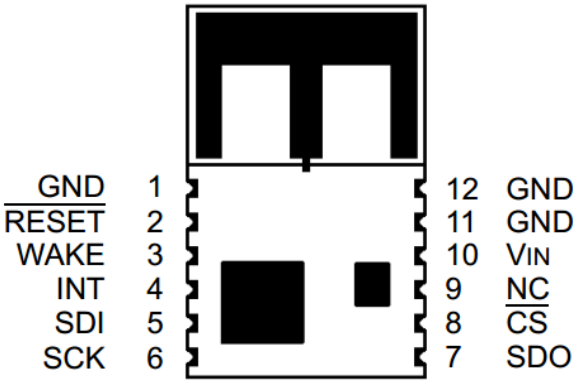


Gambar 2.12 Modul *radio transceiver* MRF24J40MA

Sumber: Microchip, 2008

MRF24J40MA yang mempunyai bentuk seperti ditunjukkan pada Gambar 2.12 adalah modul *radio frequency transceiver* yang dibuat oleh Microchip Technology Inc. Beroperasi pada frekuensi 2.4 GHz dengan standar protokol *IEEE 802.15.4.* Modul MRF24J40MA mempunyai fitur dan spesifikasi sebagai berikut:

1. Beroperasi pada *ISM Band* 2.405-2.48GHz.
2. Kecepatan data 250kbps.
3. Standar *IEEE 802.15.4.*
4. Mendukung protokol *ZigBee, MiWi, P2P,* dan *propiertary Wireless Networking Protocols.*
5. Mendukung jarak hingga 121,92m.
6. Voltase operasi 2.4-3.6V (umumnya 3.3 V).
7. Konsumsi daya yang rendah (*Tx* 23mA, *Rx* 19mA*, Sleep* 2uA)



Gambar 2.13 Diagram pin MRF24J40MA

Sumber: Microchip, 2008

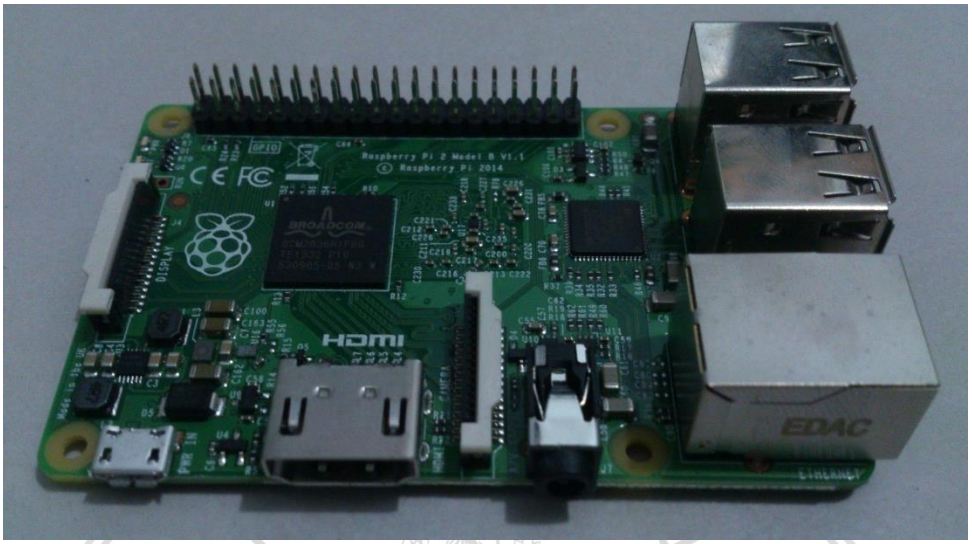
Untuk menggunakan MRF24J40MA dibutuhkan penyolderan, karena modul tersebut menggunakan teknologi *surface mount*. Juga dibutuhkan komunikasi *Serial Peripheral Interface (SPI)* dengan pin pada modul, dengan acuan diagram seperti yang ditunjukkan pada Gambar 2.13 dan Tabel 2.3.

Tabel 2.3 Fungsi *pin* pada MRF24J40MA

Pin	Simbol	Tipe	Keterangan
1	GND	Power	Digunakan sebagai *Ground.*
2	RESET	Digital Input	Digunakan untuk *reset* perangkat.
3	WAKE	Digital Input	Sebagai *trigger wake-up* eksternal.
4	INT	Digital Output	Digunakan untuk *interrupt* ke *microcontroller.*
5	SDI	Digital Input	*Serial interface data input*.
6	SCK	Digital Input	*Serial interface clock.*
7	SDO	Digital Output	Serial interface data output dari *mcu MRF24J40.*
8	CS	Digital Input	*Serial interface enable, chip select*, atau *select slave.*
9	NC	-	Tidak ada koneksi.
10	Vin	Power	Suplai daya.
11	GND	Ground	*Ground.*
12	GND	Ground	*Ground.*

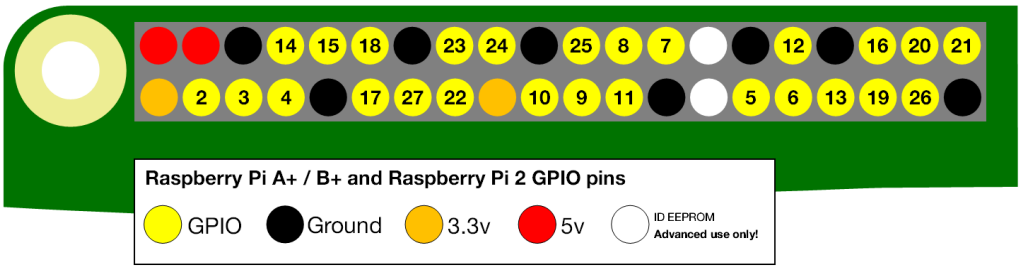
2.2.7 Raspberry Pi

Raspberry Pi adalah sebuah perangkat *single-board* komputer yang memiliki besar seukuran kartu kredit, dikembangkan di Inggris oleh Raspberry Pi Foundation. Pengembangan Raspberry Pi bertujuan sebagai bentuk pembelajaran ilmu komputer dasar pada sekolah (Sandeep, et al., 2015).



Gambar 2.14 Bentuk mikro komputer Raspberry Pi 2

Raspberry Pi 2 merupakan generasi kedua dari Raspberry Pi, Raspberry Pi 2 memiliki gambar fisik seperti yang ditujukkan oleh Gambar 2.14. Dibandingkan dengan seri sebelumnya yaitu Raspberry Pi Model B+, generasi kedua memiliki kelebihan yaitu memakai *CPU* ARM Cortex-A7 dengan 900MHz dan memiliki kapasitas *RAM* sebesar 1GB. Salah satu fitur yang terdapat pada Raspberry Pi adalah pin *GPIO (General Purpose Input/Output).*



Gambar 2.15 Pin *GPIO* Raspberry Pi

Sumber: *Raspberry Pi*, 2015

Gambar 2.15 menunjukkan penomoran pin pada Raspberry Pi. Pin tersebut merupakan penghubung Raspberry Pi dengan dunia fisik. Dalam Raspberry Pi terdapat 40 pin, 26 adalah pin *GPIO* dan sisanya adalah sumber tegangan atau *ground* (Raspberry Pi, 2015).

2.2.8 Raspberry Pi *Device Tree*

Device tree adalah struktur data yang menyerupai pohon dengan *node* (*device node*) yang mendeskripsikan sebuah perangkat keras dalam suatu sistem (Petazzoni, 2016). *Device tree* ditulis dalam bentuk teks yang disebut dengan *device tree source* (*DTS*) dan disimpan dalam file dengan format atau akhiran *dts.* Perangkat *System on a Chip* (*SoC*) modern saat ini sangatlah kompleks, *device tree* untuk sistem ini bisa memerlukan hingga ratusan baris kode. Hal ini sama dengan Raspberry Pi, yang mendukung banyak perangkat dan aksesoris tambahan, hal ini akan menimbulkan suatu masalah. Karena setiap konfigurasi yang memungkinkan dari perangkat tambahan tersebut diperlukan *device tree* yang terpisah. Hal ini bias dilakukan secara mudah dengan menggunakan *partial device tree* atau *device tree overlay. Device tree overlay* terdiri dari sejumlah *fragment*, yang masing-masing menargetkan satu *node* dan *subnode*-nya. *Device tree* ini penulis gunakan untuk mendeskripsikan perangkat MRF24J40MA pada Raspberry Pi. Tabel 2.4 menunjukkan contoh dari *device tree overlay.*

Tabel 2.4 Contoh kode program *device tree overlay*

```
1  // Enable the i2s interface
2  /dts-v1/;
3  /plugin/;
4  / {
5     compatible = "brcm,bcm2708";
6     fragment@0 {
7          target = <&i2s>;
8          __overlay__ {
9               status = "okay";
10         };
11     };
12 };
```

2.2.9 *NoSQL Database*

Istilah *Not-only SQL* (*NoSQL*) mengacu kepada mekanisme pengelolaan *data* yang tidak menggunakan model relasi tabel seperti pada *SQL* (Moniruzzaman & Hossain, 2013). Kelemahan dari model relasi tabel adalah sulitnya untuk melakukan skalabilitas secara horisontal, karena model tersebut dibangun untuk mendukung konsistensi dan ketersediaan data, sehingga ketika semakin banyak data yang ditambahkan semakin banyak pula sumberdaya yang dibutuhkan (Veen, et al., 2012). Ada dua kecenderungan dalam pertumbuhan *data* yang

mengakibatkan masalah skalabilitas ini menjadi perhatian, yaitu (Moniruzzaman & Hossain, 2013):

1. Pertumbuhan eksponensial dari jumlah *data* yang dihasilkan oleh pengguna, sistem, dan sensor. Sebagian besar *data* tersebut terletak pada penyedia sistem terdistribusi besar seperti Google, Amazon, dan penyedia layanan *cloud* lainnya.
2. Meningkatnya ketergantungan dan kompleksitas data. Hal ini dipengaruhi dengan adanya *internet*, *Web 2.0,* media sosial, dan lainnya.

BAB 3 METODOLOGI

Bab ini membahas konsep teoritik berbagai metode yang dipilih sebagai metode yang digunakan dalam penelitian.

3.1 Metode penelitian

Metode penelitian berisi prosedur atau tahapan yang ditempuh agar sistem yang dibangun dapat berjalan pada jaringan *6LoWPAN*. Jika perangkat dapat berjalan dengan baik dan menggunakan *transport layer* yang sesuai, dapat dikatakan sistem ini telah berjalan pada jaringan *6LoWPAN*.

3.1.1 Studi Literatur

Dalam perancangan penelitian, perlu dilakukan studi literatur yang menjelaskan dasar teori yang digunakan sebagai penunjang dan pendukung untuk pengembangan sistem. Adapun yang digunakan sebagai bahan studi literatur adalah sebagai berikut:

1. Penghubungan antar perangkat dengan pin *Serial Peripheral Interface (SPI).*
2. *Pyton client – server architecture.*
3. *Python IPv6 multicast.*
4. *MongoDB query.*

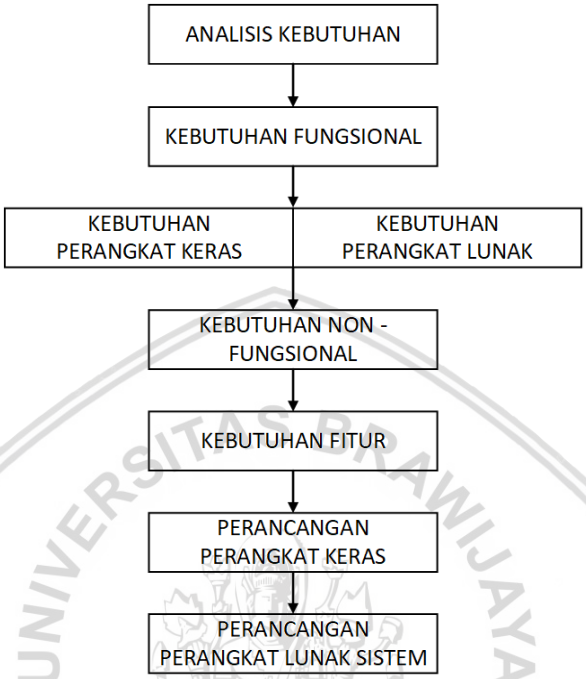
3.1.2 Identifikasi dan analisis kebutuhan sistem

Analisis kebutuhan sistem bertujuan untuk mendapatkan semua kebutuhan yang diperlukan oleh sistem yang akan dibangun dan diuji. Analisis kebutuhan dilakukan dengan mengidentifikasi kebutuhan sistem dan apa saja yang terlibat didalamnya. Dalam kebutuhan sistem akan dijabarkan proses identifikasi kebutuhan fungsional dan non-fungsional. Dalam kebutuhan fungsional akan dijabarkan mengenai identifikasi fungsi yang akan dimiliki dan berjalan pada sistem. Sedangkan dalam kebutuhan non-fungsional akan dijabarkan mengenai perangkat keras dan lunak yang dibutuhkan atau akan digunakan pada penelitian dan sistem. Dengan adanya pengidentifikasian ini akan dapat membantu serta mempermudah dalam proses desain hingga implementasi sistem.

3.1.3 Perancangan Sistem

Setelah dilakukan analisis kebutuhan sistem, maka langkah selanjutnya adalah perancangan sistem. Perancangan sistem ini berdasarkan dari proses identifikasi kebutuhan sistem pada sub-bab sebelumnya. Mengacu pada Gambar 3.1, tahap awal yang dilakukan adalah menganalisa kebutuhan non-fungsional, yaitu kebutuhan perangkat keras untuk sistem dan kebutuhan perangkat lunak untuk pengembangan sistem. Setelah diketahui kebutuhan non-fungsional, selanjutnya adalah melakukan analisa kebutuhan fungsional dari sistem.

Kebutuhan fungsional ini meliputi kemampuan apa saja yang seharusnya dimiliki oleh sistem. Dengan adanya analisa kebutuhan fungsional ini akan mempermudah tahapan selanjutnya yang mengenai perancangan perangkat. Selanjutnya akan dilakukan perancangan perangkat keras dan lunak berdasarkan dari kebutuhan fungsional yang telah dianalisa.



Gambar 3.1 Perancangan sistem keseluruhan secara umum

3.1.4 Implementasi

Implementasi sistem ini akan dilakukan sesuai dengan perancangan sistem yang telah dibuat sebelumnya. Pembahasan mengenai implementasi ini akan membahas bagaimana cara yang penulis lakukan untuk memenuhi rumusan masalah pada bab sebelumnya dan identifikasi kebutuhan yang akan dibahas pada bab selanjutnya. Implementasi yang dilakukan akan berkutat pada implementasi perangkat keras dan perangkat lunak, juga akan dibahas mengenai konfigurasi yang dilakukan.

3.1.5 Pengujian

Pada bagian pengujian akan dilakukan uji fungsional yang terdiri dari beberapa tahap. Pengujian ini dilakukan untuk dapat menunjukkan bahwa sistem yang dibuat telah mampu bekerja dengan baik sesuai spesifikasi kebutuhan yang melandasinya. Secara umum pengujian yang dilakukan adalah mengenai rumusan masalah yang telah dibuat sebelumnya. Pengujian dilakukan untuk mengetahui apakah masalah yang dirumuskan telah dapat diatasi.

Terdapat beberapa pengujian yang dilakukan dalam penelitian ini, yaitu:

1. Pengujian fungsi MRF24J40MA sebagai *6LoWPAN radio.*

2. Pengujian fungsi Raspberry Pi sebagai *6LoWPAN Router.*
3. Pengujian fitur beacon.
4. Pengujian fungsi keseluruhan sistem sebagai purwarupa *wireless sensor node* menggunakan *6LoWPAN.*

3.1.6 Analisis

Untuk mengukur kinerja dari Implementasi *6LoWPAN* Pada Purwarupa *Wireless Sensor Node*, setelah mengetahui hasil pengujian, dilakukan analisis untuk mengetahui hasil yang akan digunakan untuk menarik kesimpulan dari penelitian yang dilakukan. Hasil analisa digunakan untuk mengetahui kelayakan dari sistem yang dibuat.

3.1.7 Kesimpulan dan Saran

Pengambilan hasil dilakukan setelah semua tahapan perancangan, implementasi, dan pengujian sistem telah dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap akhir pada penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan yang terjadi dan menyempurnakan penulisan serta memberikan pertimbangan atas hasil yang telah dilakukan.

BAB 4 REKAYASA KEBUTUHAN SISTEM

Pada bab ini, dilakukan analisis mengenai kebutuhan sistem dari identifikasi yang telah dilakukan pada bab dasar teori sebelumnya. Hal ini dilakukan untuk menunjang penelitian agar hasil yang didapatkan sesuai dengan yang tujuan dari penelitian.

4.1 Identifikasi kebutuhan sistem

Pada sub-bab ini akan dilakukan proses identifikasi kebutuhan dari sistem. Proses identifikasi ini bertujuan untuk mengetahui apa saja fitur yang dibutuhkan dan atau diberikan pada sistem. Secara lengkap kebutuhan dari sistem yang akan dikembangkan adalah sebagai berikut:

1. Sistem mampu berkomunikasi menggunakan protokol *6LoWPAN.*
2. Dalam sistem terdapat sebuah *router* yang mampu memberikan alamat *IPv6* secara *stateless autoconfiguration.*
3. Pemberian fitur yang memungkinkan *node* bisa mengetahui alamat pengiriman data ke *server* secara otomatis atau tanpa konfigurasi manual, yang selanjutnya akan disebut sebagai fitur beacon.
4. Sistem mampu menggunakan *database NoSQL* untuk menyimpan data.

4.2 Analisis kebutuhan sistem

Dalam penelitian ini, ada beberapa fitur yang ingin dicapai oleh penulis, sehingga sistem ini secara keseluruhan dapat berjalan dengan baik. Secara umum tujuan yang dicapai dari sistem ini adalah pengiriman *data* dari *sensor node* menggunakan media *IPv6,* dalam hal ini adalah *6LoWPAN,* menuju *server.* Fitur yang pertama adalah sistem mampu menggunakan *6LoWPAN* sebagai media komunikasi. Pada *kernel Linux*, atau dalam hal ini sistem operasi dari Raspberry Pi yaitu *raspbian* telah diimplementasikan *6LoWPAN.* Oleh karena itu untuk menggunakan *6LoWPAN* pada Raspberry Pi, yang diperlukan adalah *modul radio* yang mendukung penggunaan standar *IEEE 802.15.4*, dalam hal ini akan digunakan MRF24J40MA. Agar Raspberry Pi bisa mengenali MRF24J40MA diperlukan sebuah *driver*. *Driver* yang dimaksud adalah *Device Tree* yang digunakan untuk mengelola alokasi sumber daya dan pemuatan *modul* secara *default.*

Selanjutnya adalah mengenai kebutuhan fitur pada *application layer*, yang dalam hal ini adalah perangkat lunak *server* dan *node*. Bahasa pemrograman yang digunakan untuk mengembangkan perangkat lunak ini adalah *Python.* Raspberry Pi mempunyai minimal satu *network interface* secara *default*, yaitu *ethernet,* atau dua dengan *WiFi* jika menggunakan Raspberry Pi versi 3. Ketika dipasangkan modul *radio* MRF24J40MA maka jumlah *network interface* juga akan bertambah. Dalam penelitian ini antarmuka jaringan yang digunakan hanyalah MRF24J40MA, untuk itu perangkat lunak yang dikembangkan hanya

akan melakukan komunikasi menggunakan antarmuka tersebut. Dalam *Python* terdapat beberapa cara untuk mencapai tujuan tersebut, menggunakan algoritma sendiri atau menggunakan *library.*

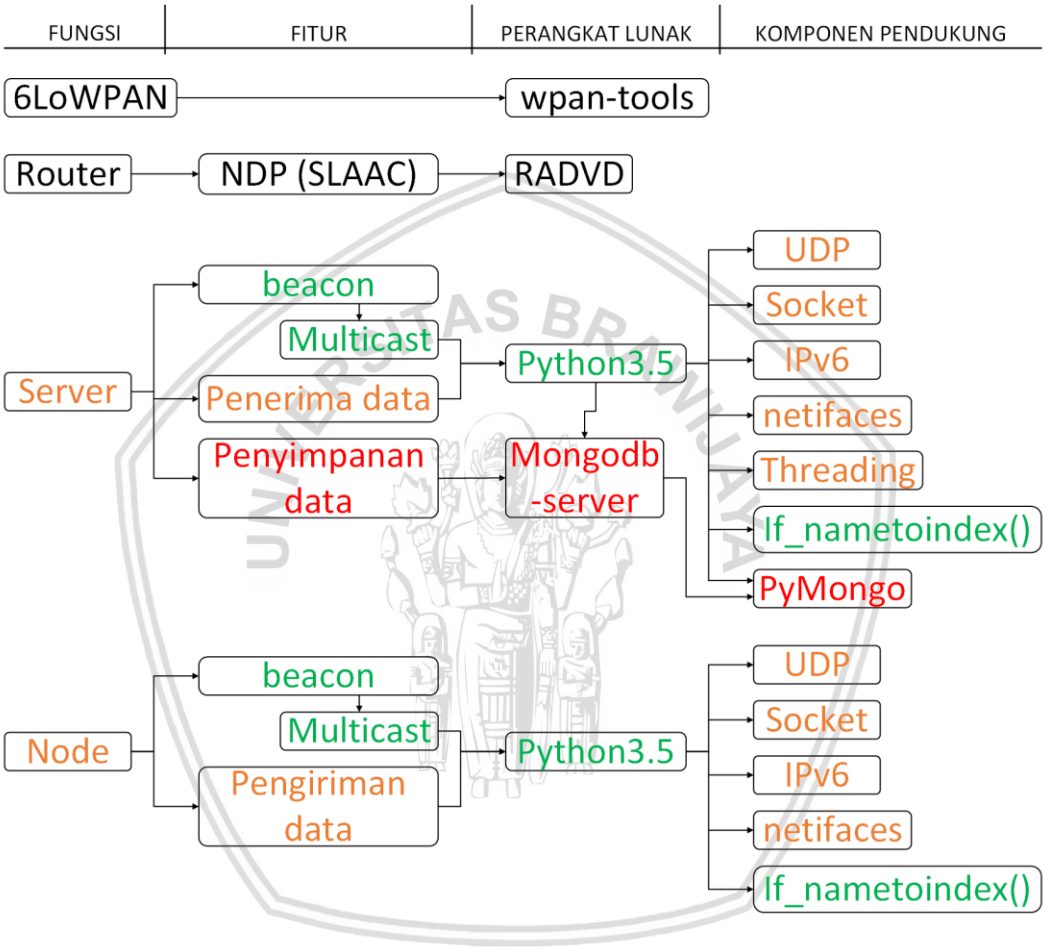
Pada penelitian ini akan digunakan *library netifaces* untuk mencapai tujuan yang telah dibahas sebelumnya. Selain berjalan dengan antarmuka yang spesifik, perankat lunak juga akan dikembangkan dengan kemampuan untuk menggunakan alamat *IP* sesuai dengan kata kunci yang diberikan. Jadi, secara umumnya adalah sebagai berikut, perangkat lunak baik itu *server* atau *node* akan diberikan sebuah kata kunci, misalnya "global". Kata kunci "global" ini merujuk pada alamat *IPv6 extended address* yang dimiliki oleh MRF24J40MA. Dan perangkat lunak hanya akan menggunakan alamat *IP* yang sesuai dengan kata kunci tersebut. Ketika dilakukan perubahan alamat *IP* oleh pengguna, perangkat lunak akan menyesuaikan dengan kata kunci yang sudah diberikan, tanpa perlu merubah kode dari perangkat lunak. Dalam perangkat lunak *server* dan *node*, protokol transportasi yang digunakan adalah *User Datagram Protocol* (*UDP*), karena secara umum komunikasi yang dilakukan adalah satu arah, dari *node* menuju *server* maka diperlukan protokol yang secara umum tidak terlalu rumit atau terlalu banyak menggunakan sumber daya.

Kebutuhan fitur yang ketiga adalah mengenai penyimpanan *data* pada *server*. Untuk kebutuhan ini akan diimplementasikan *NoSQL*, dengan menggunakan *MongoDB*. Penggunaan *MongoDB* ini dipilih karena kemudahan manipulasi *data* dan penggunaan *format file JavaScript Object Notation* (*JSON*). Dan *format data* yang digunakan ketika pertukaran *data* pada perangkat lunak yang dikembangkan ini juga menggunakan *JSON*.

Selanjutnya, yang keempat adalah mengenai fitur *beacon*. Singkatnya, fitur ini adalah ketika ada *node* yang aktif yang pertama kali dilakukan setelah mendapatkan *extended address* dari *router* adalah mencari siapa yang menjadi *server,* atau lebih tepatnya mencari alamat *IP* dari *server*. Mekanisme yang digunakan pada penelitian ini adalah *server* menginformasikan keberadaannya pada *node*, bukan sebaliknya dimana *node* yang mencari. *Server* secara periodik akan mengirim *beacon* melalui *multicast* untuk menginformasikan keberadaannya, *beacon* ini untuk saat ini hanya akan berisi alamat dari *server* yang digunakan untuk penerimaan *data*. Hal tersebut dilakukan karena mengingat topologi jaringan yang digunakan adalah *star*. Ketika *node* yang melakukan pencarian, dalam hal ini adalah melakukan pencarian secara *multicast* kepada seluruh anggota dari grup *multicast* tersebut, tentu saja semua anggota akan "mendengar" kiriman tersebut. Hal tersebut tidak diperlukan, karena *entitas* yang tidak berkepentingan tidak perlu atau tidak ada urusan dengan "mendengar" pencarian dari *node* tersebut. Maka, untuk mengetahui keberadaan *server*, *node* hanya perlu menunggu *beacon* yang dikirim oleh *server.*

Kebutuhan selanjutnya adalah mengenai pemrosesan secara paralel. Pada sisi *server* ada tiga proses yang akan mungkin berjalan bersamaan, penerimaan *data*, pemrosesan dan penyimpanan *data*, dan *beacon.* Proses *beacon* ini pasti akan tumpang tindih dengan lainnya karena akan dijalankan secara terus-menerus.

Proses penerimaan *data* juga akan sangat sibuk dengan banyaknya *data* yang datang, sistem akan membutuhkan waktu untuk melakukan penyimpanan disela-sela penerimaan ini. Untuk itu diperlukan penggunaan *multithreading* yang akan memanfaatkan *thread* yang tersedia pada *processor*. Masing-masing proses yang telah disebutkan tadi akan ditangani oleh *thread* tersendiri. Kebutuhan ini hanya berlaku pada *server*, karena *node* tidak membutuhkan pemrosesan secara paralel. Gambar 4.1 menunjukkan mengenai fitur yang akan dicapai beserta perangkat lunak dan komponen pendukung yang dibutuhkan pada fitur tersebut.



Gambar 4.1 Diagram kebutuhan sistem

4.2.1 Kebutuhan Fungsional

Dalam purwarupa sistem *wireless sensor* ini, ada beberapa kebutuhan sistem yang harus dipenuhi, sehingga purwarupa sistem ini dapat berjalan dengan baik pada lingkungan jaringan *6LoWPAN.* kebutuhan tersebut akan dijabarkan sebagai berikut:

1. Raspberry Pi bisa menggunakan MRF24J40MA sebagai *radio transceiver pada sisi router* dan *server.*
2. *Router* bisa menggunakan *neighbour discovery protocol* (*NDP*) untuk memberi *IPv6 extended address* dengan metode *stateless address autoconfiguration* (*SLAAC*) kepada *node.*

3. *Node* bisa mengirim data ke server menggunakan protokol komunikasi *6LoWPAN.*
4. *Server* bisa menerima *join request* melalui *multicast* dari *node* dan membalas pesan melalui *IPv6 short address.*
5. *Server* bisa menggunakan *database MySQL* sebagai penyimpanan data.

4.2.2 Kebutuhan Non-Fungsional

Karakteristik pengguna dalam sistem ini adalah perangkat keras *node* akan mengambil *data* dari *sensor* yang selanjutnya dikirim ke *server* menggunakan protokol komunikasi 6LoWPAN.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab ini membahas proses penerapan berbagai metode dan prinsip yang bertujuan untuk mendefinisikan sistem yang dibangun. Pada tahap ini, dilakukan perancangan sistem purwarupa *wireless sensor node* serta implementasi protokol *6LoWPAN* pada sistem.

5.1 Perancangan

Pada sub-bab ini akan dibahas mengenai perancangan sistem. Perancangan ini akan terbagi menjadi dua, perangkat keras dan perangkat lunak. Selanjutnya, perancangan perangkat lunak juga akan terbagi menjadi beberapa bagian yang akan membahas perancangan dari setiap fitur. Pembagian tersebut adalah sebagai berikut:

1. Perancangan perangkat keras.

   Sub-bab ini akan membahas mengenai cara menyambungkan Raspberry Pi dengan modul *radio* MRF24J40MA yang akan digunakan.

2. Perancangan perangkat lunak.

   a. Implementasi *6LoWPAN* pada Raspberry Pi.

      Dalam sub-bab ini akan dijabarkan mengenai metode implementasi *6LoWPAN* pada Raspberry Pi, dan cara agar modul *radio* MRF24J40MA dapat dikenali oleh Raspberry Pi.

   b. Implementasi *6LoWPAN router* pada Raspberry Pi.

      Pada sub-bab ini akan dibahas mengenai rancangan dari *router.*

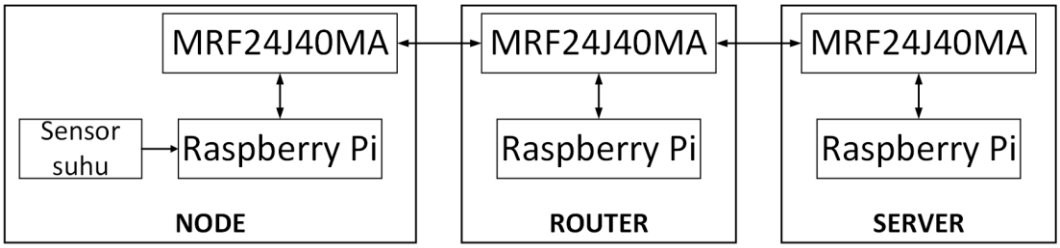
   c. Perancangan perangkat lunak *server.*

      Sub-bab ini akan membahas mengenai perancangan dari perangkat lunak dengan fungsi *server* beserta semua fitur yang akan diberikan.

   d. Perancangan perangkat lunak *node.*

      Sub-bab ini akan membahas mengenai perancangan dari perangkat lunak dengan fungsi *node* beserta semua fiturnya.

5.1.2 Perancangan sistem



Gambar 5.1 Diagram blok sistem

Secara umum, arsitektur dari sistem yang dibuat adalah seperti yang ditunjukkan pada Gambar 5.1. Sistem akan dibagi menjadi 3 bagian yang sesuai dengan peran atau tugasnya. *Node* sebagai pengirim data, *router* sebagai perangkat yang meneruskan data secara, dan server sebagai penerima dan penyimpan data. Meskipun mempunyai peran yang berbeda, ketiganya terdiri dari perangkat keras yang sama, yaitu Raspberry Pi sebagai *control unit* dan modul MRF24J40MA sebagai *radio transceiver*. Pada bagian *node* akan diberikan sensor suhu sebagai sumber data.

5.1.3 Perancangan perangkat keras

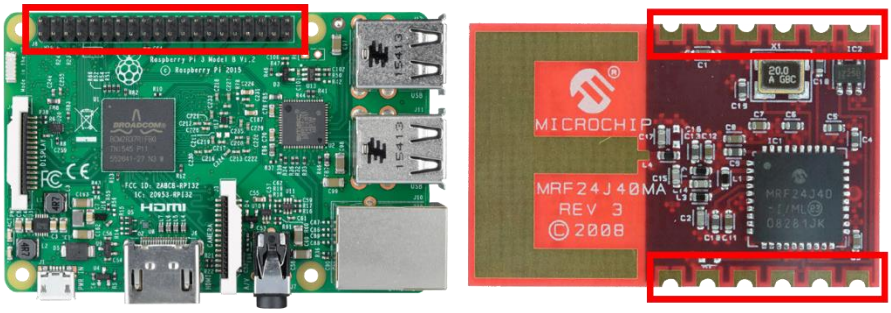
Perangkat keras yang perlu dirancang adalah penyambungan Raspberry Pi dengan modul radio MRF24J40MA. Untuk menyambungkan dua perangkat tersebut diperlukan referensi pada *datasheet* perangkat masing-masing untuk mengetahui fungsi dari masing-masing *pinout* yang ada. Setelah dilakukan studi literatur pada *datasheet* perangkat keras, dapat diketahui bahwa kegunaan dari masing-masing *pinout* dan cara menyambungkannya adalah seperti yang ditunjukkan pada Tabel 5.1.

Tabel 5.1 *Pinout* Raspberry Pi dan MRF24J40MA

Raspberry Pi		MRF24J40MA	
No. *Physical Pinout*	Keterangan	No. *Physical Pinout*	Keterangan
17	3.3V	10	Vin
19	SPIO MOSI	5	SDI
21	SPIO MISO	7	SDO
23	SPIO SCLK	6	SCK
20	GND	11	GND
22	GPIO25	4	INT
24	SPI0 CE0	8	CS

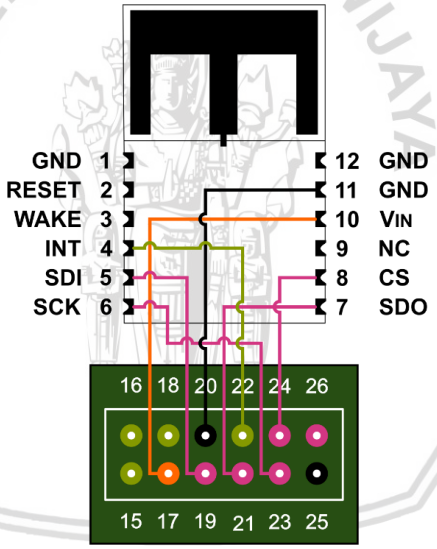
Pembahasan lebih lanjut dari masing-masing *pinout* tersebut adalah sebagai berikut:

1. *Pinout* 10 pada MRF24J40MA adalah sebagai suplai daya. Sesuai dengan spesifikasi dari pabrikan daya yang dibutuhkan atau digunakan oleh modul ini adalah 2.4 - 3.6V, sehingga *pinout* ini akan disambungkan dengan *pinout* 17 pada Raspberry Pi yang digunakan untuk suplai daya yang sama.
2. *Pinout* 5 pada MRF24J40MA adalah sebagai *serial interface data input*. *Pinout* ini digunakan sebagai *digital input*. *Pinout* ini akan disambungkan dengan *pinout* 19 pada Raspberry Pi yang digunakan sebagai *SPI MOSI* (*Master Out/ Slave In*). Dimana Raspberry Pi adalah sebagai perangkat *Master* dan MRF24J40MA sebagai perangkat *slave.*
3. *Pinout* 7 pada MRF24J40MA adalah sebagai *serial interface data output*. *Pinout* ini digunakan sebagai *digital output. Pinout* ini akan disambungkan dengan *pinout* 21 pada Raspberry Pi yang digunakan sebagai *SPI MISO* (*Master In/ Slave Out*). Keterangan dari *pinout* ini adalah kebalikan dari poin 2 sebelumnya.
4. *Pinout* 6 pada MRF24J40MA adalah sebagai *serial interface clock. Serial interface clock* ini digunakan sebagai sinkronisasi antara perangkat *master* dan *slave*. Karena *SPI* merupakan *synchronous data bus*, maka jalur untuk data dan "waktu" dibuat terpisah. (Srinivas, et al., n.d.). maka dari itu *pinout* ini akan disambungkan dengan *pinout* yang sesuai pada Raspberry Pi yaitu *pinout* 23.
5. *Pinout* 11 pada MRF24J40MA akan disambungkan dengan *pinout* 20 pada Raspberry Pi yang sama kegunaannya sebagai *ground.*
6. *Pinout* 4 pada MRF24J40MA adalah sebagai *interrupt*. *Interrupt* ini digunakan ketika perangkat *slave* memerlukan layanan dari *microprocessor* perangkat *master* dengan segera (Kumar, 2013). *Pinout* ini bisa disambungkan dengan *pinout gpio* pada Raspberry Pi, dalam hal ini akan disambungkan pada *pinout* 22 yang digunakan sebagai *gpio 25.*
7. *Pinout* 8 pada MRF24J40MA adalah sebagai *chip select* atau bisa disebut juga *slave select*. *Pinout* ini digunakan untuk menentukan perangkat *slave* mana yang digunakan. *Pinout* ini akan disambungkan dengan *pinout* 24 pada Raspberry Pi yang digunakan sebagai *spi0 ce0.*
8. *Pinout* 9 pada MRF24J40MA diberi keterangan *no connection* oleh *microchip*. Jadi *pinout* ini tidak akan digunakan.



Gambar 5.2 Perbandingan bentuk *pin* MRF24J40MA dan Raspberry Pi

Selanjutnya, penulis merasa diperlukan pembuatan skema elektronika yang selanjutnya diimplementasikan pada *pcb*. Pembuatan *pcb* ini dilakukan karena bentuk *pinout* dari MRF24J40MA adalah *smd*, sedangkan *pinout* dari Raspberry Pi adalah *male header*, perbandingan kedua *pinout* bisa dilihat pada Gambar 5.2. Nantinya, *pin* MRF24J40MA akan disolder dengan *male header* yang selanjutnya akan dihubungkan dengan sebuah *pcb,* dengan skema pada Gambar 5.3. Pada *pcb* tersebut juga akan terdapat *female header* yang akan ditancapkan pada *male header pin* Raspberry Pi untuk menghubungkan kedua perangkat.

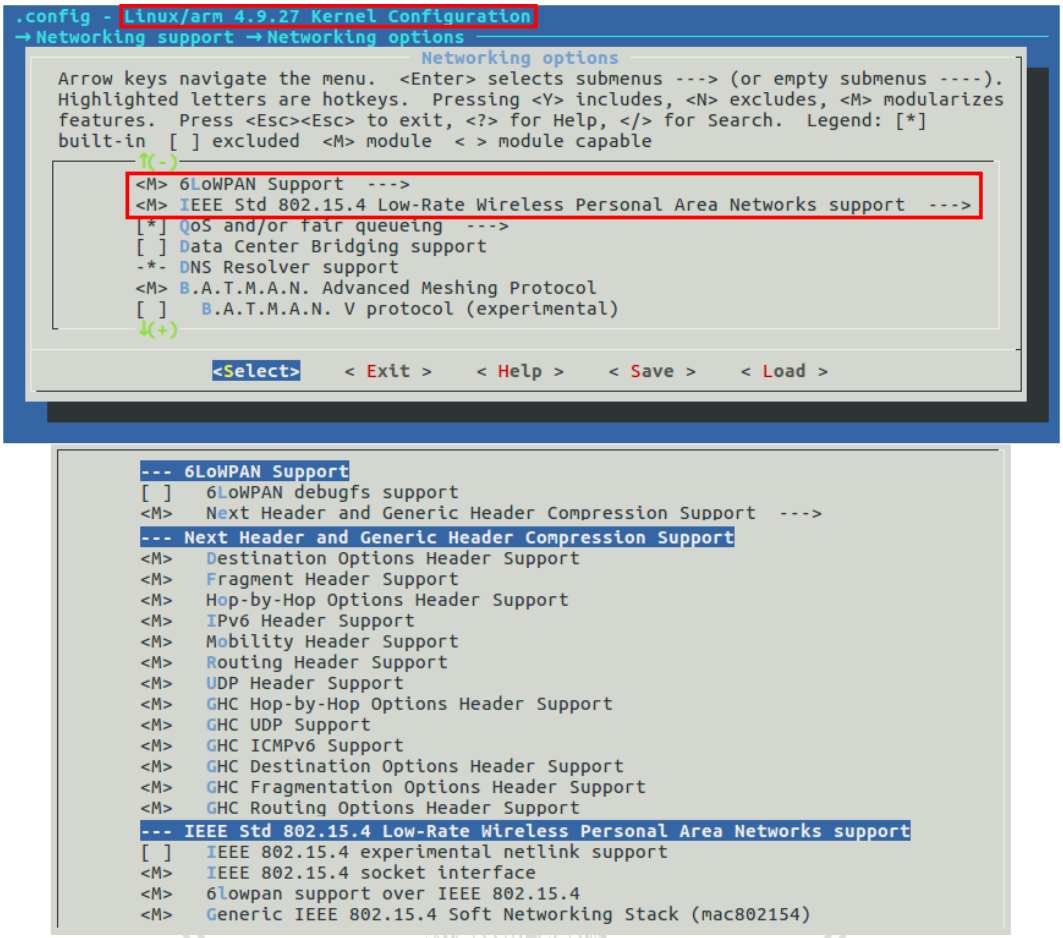


Gambar 5.3 Skema pemasangan MRF24J40MA ke Raspberry Pi

5.1.4 Perancangan implementasi *6LoWPAN* dan *router*

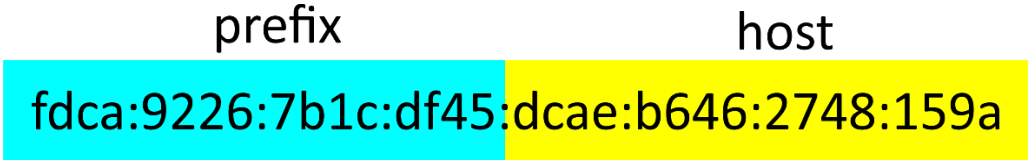
Sistem operasi yang akan digunakan adalah Raspbian dengan versi *kernel* 4.9.27. Dalam versi *kernel* tersebut, sudah didukung implementasi dari *802.15.4* dan *6LoWPAN*. Tetapi implementasi tersebut masuk dalam *kernel space,* seperti yang ditunjukkan pada Gambar 5.4. Dalam sistem operasi Linux, *system memory* dibagi kedalam dua bagian atau area, yaitu *kernel space* dan *user space*. *Kernel space* adalah lokasi dimana kode sumber dan *module kernel* berada dan dijalankan. Sedangkan *user space* adalah tempat dimana proses *normal user* berada dan dijalankan. Kedua lokasi ini mempunyai alamat memorinya sendiri. Proses dari *user space* bisa mengakses bagian dari *kernel* melalui *system call*

(Corbet, et al., 2005). Untuk bisa mengakses perangkat keras yang digunakan dibutuhkan perangkat lunak yang menyediakan akses *user space*, dalam hal ini adalah *linux-wpan* dengan *tool* yang digunakan adalah *wpan-tools.*



Gambar 5.4 Dukungan *6LoWPAN* pada *kernel* Linux

Selanjutnya adalah mengenai *router.* Perangkat lunak yang akan digunakan untuk menyediakan fungsi *router* ini adalah Radvd. Radvd menyediakan fungsi *router advertisement* yang memungkinkan penggunaan metode *stateless autoconfiguration*. Ketika ada perangkat yang akan bergabung dalam jaringan, dalam hal ini *server* atau *node*, perangkat tersebut akan mengirim pesan *router solicitation* secara *multicast*. *Router* akan membalas pesan tersebut dengan *router advertisement*. Pesan *router advertisement* yang dikirim ini salah satu isinya adalah informasi mengenai *prefix*. *Prefix* yang telah ditambahkan dengan alamat *host* inilah yang akan digunakan sebagai alamat *IPv6.* Alamat *unicast* dalam *IPv6* mempunyai ukuran 128*bits*, dimana 64*bit* pertama adalah *network prefix* yang digunakan untuk *routing*, dan 64*bit interface identifier* yang merupakan alamat dari *host*.

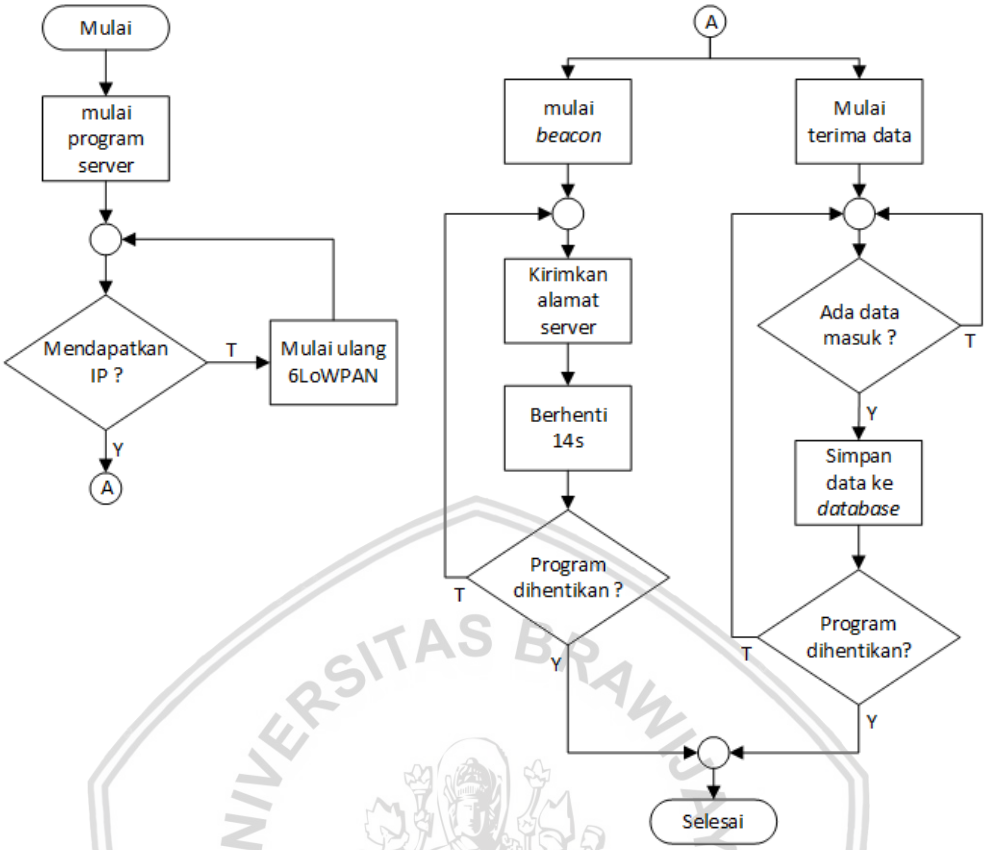


Gambar 5.5 Alamat *IPv6*

Seperti terlihat pada Gambar 5.5, dimana alamat tersebut adalah 64*bit*. Karena alamat tersebut menggunakan 64*bit* maka setengah bagian pertama, yaitu fdca:9226:7b1c:df45 adalah *prefix*, karena alamat tersebut terdiri dari 16 karakter *hexadecimal,* dan $16 \times 4 = 64$. Sedangkan sisanya dcae:b646:2748:159a adalah *host*. Karena alamat *IPv6* yang sangat panjang, dalam penulisannya alamat tersebut bisa disingkat. Jika dijabarkan, alamat lengkap dari *prefix* tersebut adalah fdca:9226:7b1c:df45:0000:0000:0000:0000/64. Setengah dari alamat *prefix* tersebut terdiri dari baris angka 0, yang bisa disingkat atau diganti menjadi "::". Sehingga alamat *prefix* bisa disingkat menjadi fdca:9226:7b1c:df45::/64. Dalam konfigurasi *Radvd* akan disertakan alamat *prefix* dan *Authoritative Border Router* (*ABRO*). *ABRO* ini dibutuhkan ketika pesan *RA* digunakan untuk menyebarluaskan *prefix* dan informasi lainnya dalam topologi *route-over* (Corbet, et al., 2005).

5.1.5 Perancangan purwarupa perangkat lunak *server*

Sesuai dengan kebutuhan sistem pada bab sebelumnya, bisa disimpulkan bahwa spesifikasi dan fitur yang akan diimplementasikan pada perangkat lunak *server* adalah menerima data dari *node* menggunakan protokol *6LoWPAN,* menggunakan *database NoSQL* sebagai penyimpanan data, dan fitur *beacon.* Ada masalah yang harus dipecahkan pada bagian *server* ini, karena terkadang data yang diterima dari *node* terjadi secara bersamaan, dan hal ini akan sangat mungkin dan bahkan sering terjadi, karena *node* akan didesain untuk mengirim datanya ketika data tersebut sudah tersedia. Fitur *beacon* juga akan menambah masalah karena rancangan dari fitur ini adalah untuk berjalan terus-menerus selama *server* aktif. Solusi dari masalah ini adalah dengan menggunakan pemrosesan parallel dengan *multiprocessing* atau secara *concurrent* menggunakan *multithreading*. Pada penelitian ini digunakan *multithreading* karena fitur yang diimplementasikan tidak membutuhkan *resource* yang terlalu banyak.



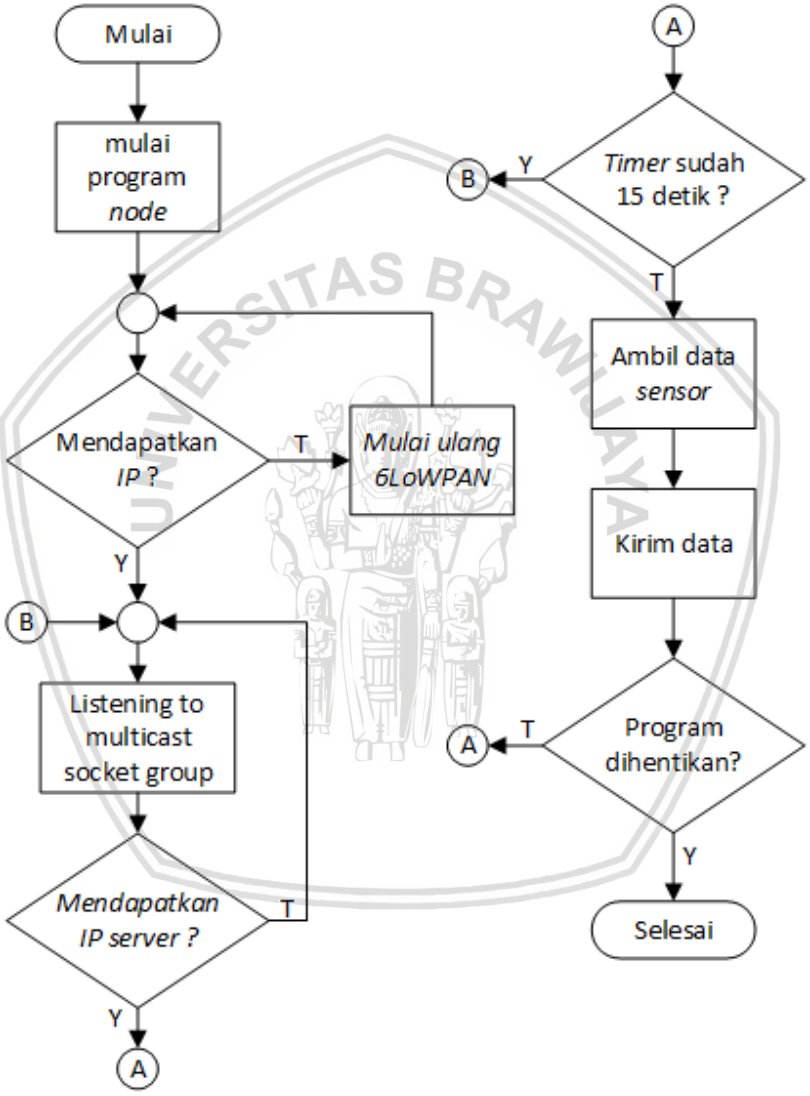
Gambar 5.6 *Flowchart* rancangan purwarupa perangkat lunak *server*

Dengan mengacu pada *flowchart* di Gambar 5.6, setelah perangkat lunak *server* dinyalakan, akan dilakukan pengecekan apakah modul radio sudah mendapatkan *global IPv6* dari *router*. Jika hanya tersedia alamat *link-local* yang artinya perangkat belum mendapatkan *IP* akan dilakukan *restart* pada *interface*, hingga mendapatkan *global IPv6*. Selanjutnya adalah menjalankan fungsi *beacon*. Hal pertama yang dilakukan adalah melakukan inisialisasi *socket multicast,* lalu menyimpan alamat *IP* global didalam sebuah variabel, yang selanjutnya isi dari variabel tersebut akan dikirimkan melalui *multicast* secara periodik dengan waktu tunggu tiga detik.

Fungsi *beacon* ini akan berjalan secara terus-menerus dan menggunakan *daemon thread*. *Daemon thread* ini digunakan karena apa yang dilakukan fungsi *beacon* adalah proses yang monoton, dimana tidak diperlukan banyak perhatian atau bisa dikatakan jalankan lalu lupakan. *Daemon thread* akan berhenti ketika tidak ada *thread* lainnya yang berjalan, dengan kata lain *thread* ini akan otomatis berhenti ketika *server* dimatikan. Setelah itu fungsi *listening* akan dijalankan, dan akan berjalan pada *main thread*. Ketika ada data yang diterima dari *node* proses pengolahan data yang termasuk penyimpanan pada *database* akan dialihkan ke *thread* lainnya, misalnya *thread-1*, sehingga *main thread* bisa melayani pengiriman data lainnya yang datang.

Implementasi *6LoWPAN* pada perangkat lunak ini adalah mengenai penggunaan MRF24J40MA sebagai media komunikasi. Untuk dapat berkomunikasi menggunakan *socket*, dalam bahasa pemrograman Python, khususnya untuk *IPv6*, dibutuhkan *tuple* yang berisi *host*, *port*, *flow info,* dan *scope id*. Untuk bisa mendapatkan alamat *IP* dari *interface* (*host*) akan digunakan *modul* Netifaces. Modul tersebut akan digunakan dalam implementasi perangkat lunak *server* dan *node*.

5.1.6 Perancangan purwarupa perangkat lunak *node*



Gambar 5.7 *Flowchart* rancangan purwarupa perangkat lunak *node*

Untuk rancangan *node*, penulis tidak akan mengimplementasikan *multithreading*. Hal ini dikarenakan proses yang dilakukan oleh *node* adalah sangat sederhana dan tidak membutuhkan pemrosesan secara parallel. Dengan Gambar 5.7 sebagai acuan, sama halnya seperti *server*, hal pertama yang dilakukan adalah mengecek apakah modul radio yang digunakan sudah mendapatkan *global IP*. Setelah proses tersebut selesai, akan dilakukan *listening*

pada *multicast group*, ini dilakukan untuk mendapatkan *multicast* dari *server* yang berisi alamat *IP* dari *server*. Setelah mengetahui alamat *IP* dari *server*, *node* akan mengambil data dari *sensor*, dalam hal ini akan dipakai nilai suhu.

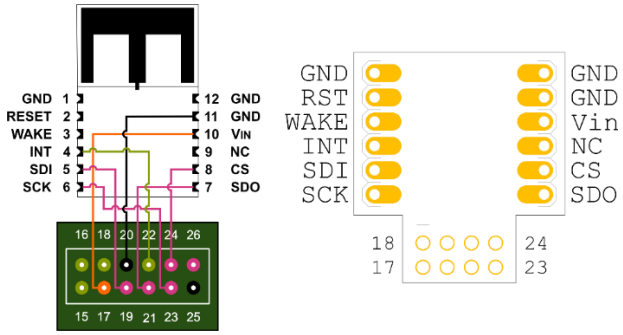
Data dari *sensor* suhu hanyalah berupa angka, yang umumnya berbentuk tipe data *float*. Untuk mereprentasikan nilai suhu ini penulis hanya memasangkan satu *sensor* suhu pada satu *node* Raspberry Pi, sedangkan *node* lainnya akan direpresentasikan menggunakan fungsi yang akan menghasilkan nilai angka secara acak. Untuk mengambil data dari *sensor* suhu akan digunakan *library* dari AdaFruit. Dan sesuai dengan anjuran dari AdaFruit, proses pengambilan data akan dilakukan dengan interval 3 detik sekali, atau dengan waktu tunggu antar pembacaan selama dua detik. Selanjutnya tentu data tersebut akan dikirimkan ke *server.*

Fungsi *beacon* yang diimplementasikan pada *node* tentu saja memiliki perbedaan dengan *server*. Selain peran yang berbeda, *node* akan diimplementasikan metode *keep-alive*. Singkatnya, ketika *node* selesai melakukan inisialisasi (cek *IP* dan *listening multicast*), setiap 15 detik sekali, node akan berhenti melakukan pembacaan dan pengiriman data untuk melakukan *listening* di *multicast group*. Hal ini dilakukan untuk mengetahui apakah alamat *IP server* masih sama dengan sebelumnya atau apakah *server* masih *online.* Metode *keep-alive* ini didesain sesuai dengan pedoman penggunaan protokol *UDP* yang disusun oleh *Internet Engineering Task Force* (*IETF*), dimana jika digunakan *keep-alive* dalam *UDP* tidak dianjurkan untuk mengirim pesan *keep-alive* lebih dari sekali dalam 15 detik (Internet Engineering Task Force (IETF), 2017). Hal tersebut dikarenakan pesan *keep-alive* pada umumnya akan memakan banyak sumberdaya, baik itu sistem atau jaringan.

5.2 Implementasi

Pada sub-bab ini akan dibahas mengenai implementasi perangkat keras dan lunak. Pembahasan yang dilakukan adalah mengenai kelanjutan dari rancangan yang telah dibuat sebelumnya.

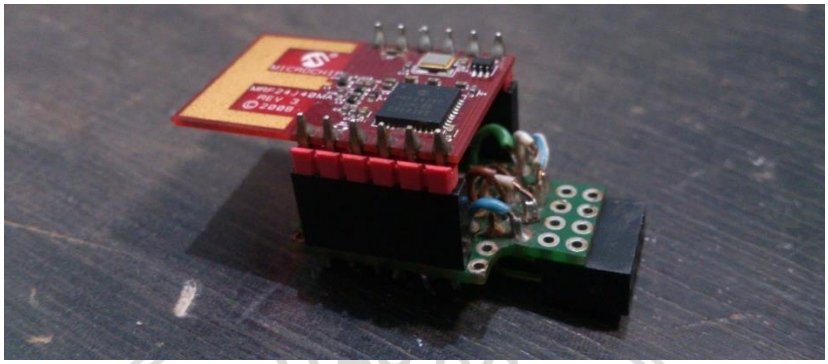
5.2.1 Menghubungkan Raspberry Pi dengan MRF24J40MA



Gambar 5.8 Skema dan *layout pcb* MRF24J40MA ke Raspberry *Pi*

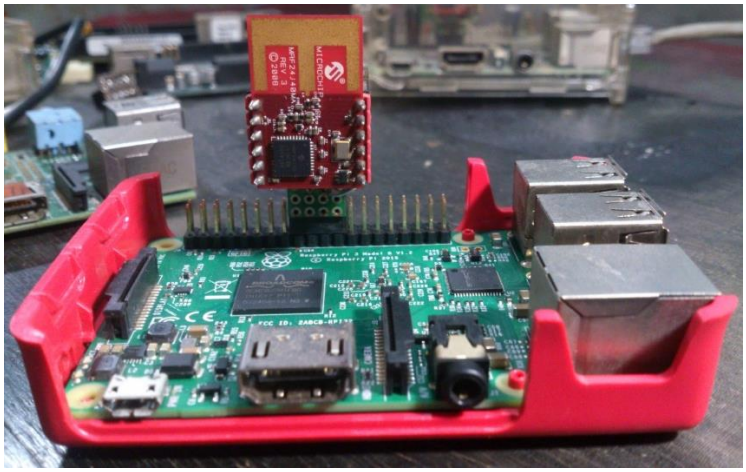
Setelah diketahui bagaimana cara menyambungkan *pinout* yang dibutuhkan pada sub-bab perancangan sebelumnya, selanjutnya penulis membuat skema

rangkaian elektronika seperti yang ditunjukkan pada Gambar 5.8. Dengan dibuatnya skema rangkaian ini akan memudahkan ketika melakukan perakitan perangkat keras, karena kita akan tahu apa saja yang dibutuhkan dan bagaimana cara merangkainya. Skema rangkaian yang penulis buat adalah dalam bentuk *layout pcb*, hal ini dilakukan karena untuk menyambungkan kedua perangkat keras tersebut hanya dibutuhkan penyambungan yang sederhana dan hasilnya bisa dilihat pada Gambar 5.9.



Gambar 5.9 MRF24J40MA terpasang dengan *pcb*

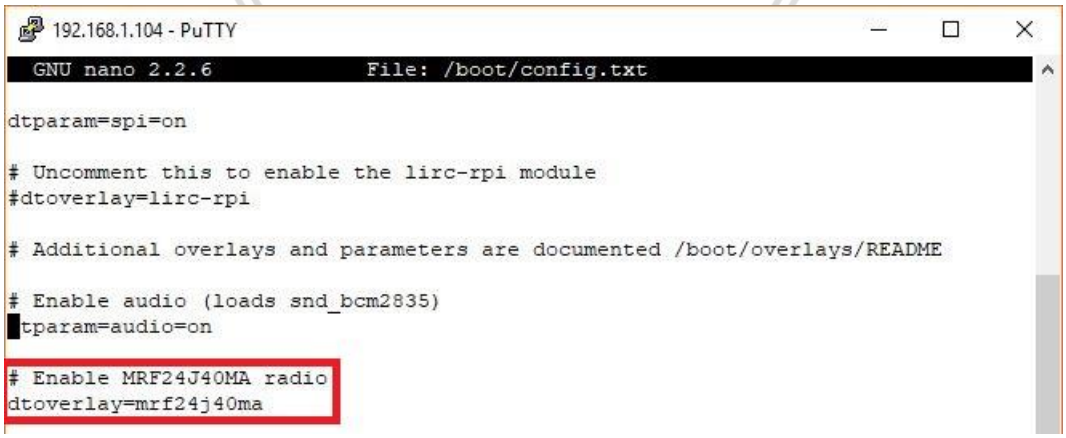
Gambar 5.10 menunjukkan bagaimana MRF24J40MA yang terpasang pada Raspberry Pi dengan posisi *pin GPIO* yang terletak pada tepat bagian tengah dari keseluruhan *pin*. Pemilihan posisi tersebut selain karena posisi spi0 yang berada ditengah, hal ini memudahkan ketika akan menambah perangkat keras yang dipasang pada Raspberry Pi. Karena masih banyak *pin* yang tidak terpakai dan *modul* MRF24J40MA tidak memakan banyak tempat.



Gambar 5.10 MRF24J40MA terpasang pada Raspberry Pi

Selanjutnya adalah mengenai konfigurasi *driver* MRF24J40MA. Daftar kode perintah di bawah adalah perintah yang dijalankan untuk melakukan pemasangan *driver*. Baris pertama adalah mengenai pemasangan perangkat lunak yang dibutuhkan untuk melakukan *compile* pada *file device tree*. Baris ke dua adalah mengenai pembuatan *file device tree* yang akan dijelaskan nanti. Baris ke tiga adalah melakukan *compile file* dtbo menjadi dts. Yang terakhir adalah memberikan komfigurasi pada *file* config.txt seperti pada Gambar 5.11.

Kode perintah pembuatan *device tree*	
1	`sudo apt-get install device-tree-compiler dh-autoreconf libnl-3-dev libnl-genl-3-dev git`
2	`sudo nano mrf24j40ma-overlay.dts`
3	`dtc -@ -O dtb -o mrf24j40ma.dtbo mrf24j40ma-overlay.dts`
4	`sudo cp mrf24j40ma.dtbo /boot/overlays/.`
5	`sudo nano /boot/config.txt`



Gambar 5.11 Konfigurasi *file config* pada sistem operasi Raspberry Pi

Kode sumber *device tree* untuk MRF24J40MA

```
1  /dts-v1/;
2  /plugin/;
3  /{
4  compatible = "bcrm,bcm2835", "bcrm,bcm2836",
5               "bcrm,bcm2708", "bcrm,bcm2709";
6  fragment@0 {
7      target = <&spi0>;
8          __overlay__ {
9                   #address-cells = <1>;
10                  #size-cells = <0>;
11                  status = "okay";
12                  mrf24j40@0 {
13                          compatible = "mrf24j40";
14                          reg = <0>;
15                          interrupts = <23 8>;
16                          interrupt-parent = <&gpio>;
17                          spi-max-frequency = <5000000>;
18                          };
19                  spidev@0 { status = "disabled"; };
20                  spidev@1 { status = "disabled"; };
21                  };
22              };
23          };
```

Tabel di atas adalah kode sumber untuk *file device tree*. Baris ke empat adalah untuk menyatakan versi *microcontroller*, dalam hal ini yang digunakan oleh Raspberry Pi, yang bisa menggunakan *file device tree* ini. Baris ke tujuh menyatakan target dari *device tree* ini adalah untuk *spi0*. Baris ke 12 sampai 17 adalah konfigurasi untuk MRF24J40MA. Baris ke 14 mengenai "reg" adalah *spi* yang digunakan. Nilai "0" yang diberikan adalah karena MRF24J40MA dipasang pada *pin* 24 yang digunakan sebagai *chip select spi0*. Selanjutnya, *interrupt* pada baris 15, menyatakan lokasi *pin* dan *pin* mode yang digunakan. Angka 23 adalah *pin GPIO* pada Raspberry Pi yang tersambung dengan *pin SCK* pada MRF24J40MA. Angka 8 adalah mode *low level*, maka *interrupt* akan terpicu ketika *pin SCK* (dan *pin GPIO 23*) dalam keadaan *low*. Selanjutnya baris ke 17 mengenai frekuensi yang ditetapkan pada 5Mhz sesuai dengan spesifikasi MRF24J40MA. Untuk mengetahui apakah *device tree* bisa dijalankan oleh *kernel* dan MRF24J40MA sudah terdeteksi dan bisa digunakan, bisa dilihat melalui hasil keluaran dari *buffer kernel* yang muncul ketika proses *boot* berjalan, seperti yang ditunjukkan pada Gambar 5.12.

```
pi@raspberrypi:~/pan6$ dmesg | grep mrf
[    5.611460] mrf24j40 spi0.0: probe(). IRQ: 191
pi@raspberrypi:~/pan6$
```

Gambar 5.12 Hasil *output dmesg*

5.2.2 Implementasi *6LoWPAN*

Kode perintah pemasangan *wpan-tools*

```
1  git clone https://github.com/linux-wpan/wpan-tools
2  cd wpan-tools
3  ./autogen.sh
4  ./configure CFLAGS='-g -O0' --prefix=/usr --sysconfdir=/etc --
   libdir=/usr/lib
5  make
6  sudo make install
```

Daftar kode perintah di atas adalah mengenai pemasangan *wpan-tools* pada Raspberry Pi. Setelah melakukan pengunduhan kode sumber (baris 1) selanjutnya dilakukan *compile* seperti yang ditunjukkan pada baris 3 sampai 6. Setelah *wpan-tools* terpasang, bisa dilakukan perintah *iwpan phy* dan *iwpan dev* untuk mengetahui apakah *wpan-tools* sudah terpasang dengan baik dan bisa mengenali *interface* MRF24J40MA, seperti yang ditunjukkan pada Gambar 5.13.

```
pi@raspberrypi:~/pan6$ iwpan phy
wpan_phy phy0
supported channels:
        page 0: 11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
current_page: 0
current_channel: 11,  2405 MHz
cca_mode: (2) Carrier sense only
cca_ed_level: -69
tx_power: 0
capabilities:
        iftypes: node,monitor
        channels:
                page 0:
                        [11]  2405 MHz, [12]  2410 MHz, [13]  2415 MHz,
                        [14]  2420 MHz, [15]  2425 MHz, [16]  2430 MHz,
                        [17]  2435 MHz, [18]  2440 MHz, [19]  2445 MHz,
                        [20]  2450 MHz, [21]  2455 MHz, [22]  2460 MHz,
                        [23]  2465 MHz, [24]  2470 MHz, [25]  2475 MHz,
                        [26]  2480 MHz
pi@raspberrypi:~/pan6$ iwpan dev
phy#0
        Interface wpan0
                ifindex 3
                wpan_dev 0x1
                extended_addr 0x6e7658f5b6b24dc1
                short_addr 0x0002
                pan_id 0x0024
                type node
                max_frame_retries 3
                min_be 3
                max_be 5
                max_csma_backoffs 4
                lbt 0
                ackreq_default 0
```

Gambar 5.13 *Wpan-tools* bisa mengenali MRF24J40MA

Langkah selanjutnya adalah memberikan konfigurasi berupa alamat *IP*, *pan id* dan lainnya ke *interface*. Dalam penelitian ini, penulis memutuskan untuk membuat beberapa *file* yang berisi konfigurasi jaringan untuk MRF24J40MA. Hal

ini dilakukan untuk mempersingkat waktu proses konfigurasi ini, yang cukup memakan banyak waktu jika dilakukan satu per satu. *File* yang dibuat adalah *start_pan6* untuk memulai*, stop_pan6* untuk menghentikan, *restart_pan6* untuk memulai ulang, dan *monitor_pan6* untuk merubah *interface* menjadi mode *monitor*. MRF24J40MA tidak mempunyai alamat fisik *MAC* yang statis, maka alamat ini akan selalu berubah ketika perangkat dinyalakan ulang atau terjadi *restart*. Karena alamat *MAC* ini nantinya akan digunakan sebagai alamat *IP host*, harus diberikan alamat yang statis, karena jika tidak mala *IP* dari perangkat akan selalu berubah. Pemberian alamat *MAC* ini terlihat pada baris ke tiga dalam tabel kode perintah konfigurasi *6LoWPAN.* Baris ke empat menyatakan *channel* yang digunakan, pemilihan 11 dikarenakan *channel* tersebut merupakan salah satu *channel* yang tidak bertumpukan dengan *channel lainnya.* Konfigurasi *channel* dan *pan id* pada baris ke enam ini akan dibuat sama untuk semua perangkat.

Kode perintah konfigurasi *6LoWPAN*	
1	sudo nano start_pan6
2	#!/bin/bash
3	sudo ip link set dev wpan0 address DE:AE:B6:46:27:48:A5:9A
4	sudo iwpan phy phy0 set channel 0 11
5	sudo ip link add link wpan0 name lowpan0 type lowpan
6	sudo iwpan dev wpan0 set pan_id 0x24
7	sudo ip addr add fdca:9226:7b1c:df45:dcae:b646:2748:a59a/64 dev lowpan0
8	sudo ip link set wpan0 up
9	sudo ip link set lowpan0 up

Kode perintah untuk *stop_pan6* sederhananya hanyalah perintah *interface* untuk berhenti, seperti dalam baris 12 dan 13, lalu menghapus *interface* pada baris 14. Hal tersebut juga dilakukan dalam re*start_pan6,* bedanya pada akhir baris 20 dilakukan pemanggilan *file start_pan6* untuk memulai ulang *interface*. Perintah *monitor_pan6*, pada baris 21 sampai 27, digunakan untuk merubah *interface wpan0* menjadi *monitor0*. *Interface monitor0* ini digunakan untuk *sniffing*.

Kode perintah konfigurasi *6LoWPAN* - lanjutan	
10	sudo nano stop_pan6
11	#!/bin/bash
12	sudo ip link set wpan0 down
13	sudo ip link set lowpan0 down
14	sudo ip link del lowpan0 type lowpan
Kode perintah konfigurasi *6LoWPAN* - lanjutan	
15	sudo nano restart_pan6
16	#!/bin/bash
17	sudo ip link set wpan0 down
18	sudo ip link set lowpan0 down

```
19   sudo ip link del lowpan0 type lowpan
20   sudo /home/pi/pan6/start_pan6
```

Kode perintah konfigurasi *6LoWPAN* - lanjutan

```
21  sudo nano monitor_pan6
22   #!/bin/bash
23   sudo ip link set wpan0 down
24   sudo ip link set lowpan0 down
25   sudo ip link del lowpan0 type lowpan
26   sudo iwpan phy phy0 interface add monitor0 type monitor
27   sudo ip link set monitor0 up
```

Selanjutnya adalah pembuatan *file service* yang akan dimasukkan dalam *sysytemd* (baris 41 dan 42). Hal ini akan memungkinkan MRF24J40MA sudah siap digunakan ketika sistem dinyalakan, tanpa harus konfigurasi *interface* secara manual. Dalam pembuatan *file service* ini hanya perlu memasukkan lokasi dari *file* konfigurasi yang telah dibuat sebelumnya (*start_pan6* dan lainnya) seperti pada baris ke 35 sampai 37. Setelah itu, untuk menjalankan perintah *start, stop,* dan *restart* hanya perlu menjalankan perintah sudo service pan6 <perintah> seperti pada Gambar 5.14.

Kode perintah konfigurasi *6LoWPAN* - lanjutan

```
28  sudo nano pan6.service
29   [Unit]
30   Description=Create 6lowpan (IEEE802.15.4) network device
31   After=network.target
32   [Service]
33   Type=oneshot
34   User=root
35   ExecStart=/home/pi/pan6/start_pan6
36   ExecStop=/home/pi/pan6/stop_pan6
37   ExecRestart=/home/pi/pan6/restart_pan6
38   RemainAfterExit=yes
39   [Install]
40   WantedBy=multi-user.target
41  sudo mv pan6/pan6.service /etc/systemd/system
42  sudo systemctl enable pan6.service
```

```
pi@raspberrypi:~$ ifconfig lowpan0
lowpan0: error fetching interface information: Device not found
pi@raspberrypi:~$ sudo service pan6 start
pi@raspberrypi:~$ ifconfig lowpan0
lowpan0   Link encap:UNSPEC  HWaddr 6E-76-58-F5-B6-B2-4D-C1-00-00-00-00-00-00-00
-00
          inet6 addr: fe80::6c76:58f5:b6b2:4dc1/64 Scope:Link
          inet6 addr: fe80::24:ff:fe00:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1280  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:2405 (2.3 KiB)
```

Gambar 5.14 perintah *pan6 service*

5.2.3 Implementasi *router*

Pada pemasangan perangkat lunak Radvd, hanya diperlukan pengunduhan, seperti pada baris pertama tabel perintah pemasangan *router*, dan melakukan *compile* seperti pada baris 3 sampai 6.

Kode perintah pemasangan *router*	
1	`git clone https://github.com/linux-wpan/radvd.git -b 6lowpan`
2	`cd radvd`
3	`./autogen.sh`
4	`./configure --prefix=/usr/local --sysconfdir=/etc --mandir=/usr/share/man`
5	`make`
6	`sudo make install`

Setelah terpasang, selanjutnya dilakukan pemberian konfigurasi untuk Radvd, seperti pada baris pertama tabel kode perintah konfigurasi Radvd, denga nisi konfigurasi pada tabel kode sumber konfigurasi Radvd. Baris ke 2 dan 3 adalah mengaktifkan *forwarding* untuk *IPv6 interface,* seperti yang terlihat pada Gambar 5.15. Hal tersebut dilakukan supaya perangkat *router* bisa mengirim pesan *neighbor advertisement* dengan *flag set* sebagai *router.*

Kode perintah konfigurasi *RADVD*	
1	`sudo nano /etc/radvd.conf`
2	`sudo nano /etc/sysctl.conf`
3	`net.ipv6.conf.all.forwarding=1`
4	`radvd -c`

```
  GNU nano 2.7.4                File: /etc/sysctl.conf

#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
net.ipv6.conf.all.forwarding=1
```

Gambar 5.15 Konfigurasi *IPv6 forwarding* pada *sysctl*

Kode sumber konfigurasi *RADVD*	
1	`interface lowpan0`

```
2  {
3      AdvSendAdvert on;
4      UnicastOnly on;
5      AdvCurHopLimit 255;
6      AdvSourceLLAddress on;
7      prefix fdca:9226:7b1c:df45::/64
8      {
9          AdvOnLink off;
10         AdvAutonomous on;
11         AdvRouterAddr on;
12     };
13     abro fe80::dcae:b646:2748:a59a
14     {
15         AdvVersionLow 10;
16         AdvVersionHigh 2;
17         AdvValidLifeTime 2;
18     };
19 };
```

Keterangan kode sumber dari tabel kode sumber konfigurasi Radvd adalah sebagai berikut:

1. AdvSendAdvert on;

   Mengaktifkan pengiriman pesan *router advertisement* secara berkala dan merespon pesan *router solicitation*.

2. UnicastOnly on;

   Tipe *link interface* yang didukung atau digunakan hanyalah *unicast.* Dengan mengaktifkan pilihan ini, maka pesan *router advertisement* hanya akan dikirimkan ketika ada *node* yang meminta dan dikirim ke *node* tersebut melalui *unicast*.

3. AdvCurHopLimit 255;

   Nilai *hop limit* di *hop count* field yang terletak pada *IP* header untuk paket *IP* yang keluar diatur pada nilai 255.

4. AdvSourceLLAddress on;

   Ketika pilihan ini diaktifkan, maka alamat *link-layer* dari pengirim akan disertakan dalam pesan *router advertisement.*

5. Baris ke 7 sampai 12 adalah mengenai pengaturan dari *prefix*.

6. AdvOnLink off;

Ketika di atur menjadi *off*, maka pesan *router advertisement* tidak akan menyatakan bahwa atribut dari *prefix* adalah *on-link* atau *off-link.*

7. AdvAutonomous on;

   Pilihan ini digunakan untuk mengaktifkan *stateless autoconfiguration.*

8. AdvRouterAddr on;

   Pilihan ini akan mengaktifkan pengiriman alamat *interface* di pesan *router advertisement,* bukan alamat *prefix.*

9. Baris ke 13 sampai 17 adalah mengenai pengaturan *authoritative border router option*.

Konfigurasi untuk Radvd ini bisa dilakukan pengecekan untuk mengetahui apakah *syntax* yang digunakan sudah benar, dengan cara memberi perintah seperti pada Gambar 5.16. Radvd bisa diaktifkan dalam *debug mode* untuk mengetahui aktivitas yang terjadi dalam *router*, seperti yang terlihat pada Gambar 5.17.

```
pi@raspberrypi:~ $ radvd -c
[Jul 18 01:40:42] radvd (1007): config file, /etc/radvd.conf, syntax ok
pi@raspberrypi:~ $
```

Gambar 5.16 Memeriksa kebenaran kode konfigurasi Radvd

```
pi@raspberrypi:~ $ sudo radvd -d 5 -m stderr -n
[Jul 18 01:39:00] radvd (999): version 2.13 started
[Jul 18 01:39:00] radvd (999): lowpan0 interface definition ok
[Jul 18 01:39:00] radvd (999): config file, /etc/radvd.conf, syntax ok
[Jul 18 01:39:00] radvd (999): radvd startup PID is 999
[Jul 18 01:39:00] radvd (999): opened pid file /var/run/radvd.pid
[Jul 18 01:39:00] radvd (999): locked pid file /var/run/radvd.pid
[Jul 18 01:39:00] radvd (999): opened pid file /var/run/radvd.pid
[Jul 18 01:39:00] radvd (999): radvd PID is 999
[Jul 18 01:39:00] radvd (999): wrote pid 999 to pid file: /var/run/radvd.pid
[Jul 18 01:39:00] radvd (999): validated pid file, /var/run/radvd.pid: 999
[Jul 18 01:39:00] radvd (999): initializing privsep
[Jul 18 01:39:00] radvd (999): radvd privsep PID is 1000
[Jul 18 01:39:00] radvd (999): lowpan0 if_index changed from 0 to 5
[Jul 18 01:39:00] radvd (999): ioctl(SIOCGIFFLAGS) succeeded on lowpan0
[Jul 18 01:39:00] radvd (999): lowpan0 is up
[Jul 18 01:39:00] radvd (999): lowpan0 is running
[Jul 18 01:39:00] radvd (999): lowpan0 supports multicast
[Jul 18 01:39:00] radvd (999): lowpan0 mtu: 1280
[Jul 18 01:39:00] radvd (999): lowpan0 hardware type: ARPHRD_6LOWPAN
[Jul 18 01:39:00] radvd (999): lowpan0 link layer token length: 64
[Jul 18 01:39:00] radvd (1000): Freeing Interfaces
[Jul 18 01:39:00] radvd (1000): freeing interface lowpan0
[Jul 18 01:39:00] radvd (999): lowpan0 prefix length: 64
[Jul 18 01:39:00] radvd (999): lowpan0 linklocal address: fe80::24:ff:fe00:1
[Jul 18 01:39:00] radvd (999): lowpan0 address: fe80::24:ff:fe00:1
[Jul 18 01:39:00] radvd (999): lowpan0 address: fe80::9437:cc9:dd9a:30e6
[Jul 18 01:39:00] radvd (999): lowpan0 is ready
[Jul 18 01:39:00] radvd (999): polling for 0 second(s), next iface is lowpan0
[Jul 18 01:39:00] radvd (999): timer_handler called for lowpan0
[Jul 18 01:39:00] radvd (999): no client list, no destination, unicast only...do
ing nothing
[Jul 18 01:39:00] radvd (999): lowpan0 next scheduled RA in 16 second(s)
[Jul 18 01:39:00] radvd (999): polling for 16 second(s), next iface is lowpan0
[Jul 18 01:39:16] radvd (999): timer_handler called for lowpan0
[Jul 18 01:39:16] radvd (999): no client list, no destination, unicast only...do
ing nothing
```

Gambar 5.17 Radvd dalam *debug mode*

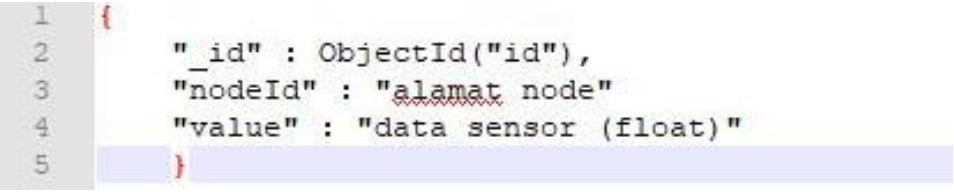
5.2.4 Implementasi purwarupa perangkat lunak *server*

Pada sisi *server* digunakan perangkat lunak Mongodb sebagai media penyimpanan data. Tetapi dari Mongodb versi terakhir yang bisa digunakan untuk Raspberry Pi adalah versi 2.4.14 seperti yang ditunjukkan pada Gambar 5.18. Ketika penlitian ini dilakukan versi terbaru dari Mongodb adalah versi 3, yang membawa banyak pembaruan dan fitur.

```
Inst libsnappy1v5 (1.1.3-3 Raspbian:stable [armhf])
Inst libv8-3.14.5 (3.14.5.8-11 Raspbian:stable [armhf])
Inst mongodb-clients (1:2.4.14-4 Raspbian:stable [armhf])
Inst mongodb-server (1:2.4.14-4 Raspbian:stable [armhf])
Conf libpcrecpp0v5 (2:8.39-3 Raspbian:stable [armhf])
Conf libboost-system1.58.0 (1.58.0+dfsg-5.1+rpi1+b1 Raspbian:stable [armhf])
```

Gambar 5.18 Versi Mongodb untuk Raspbian

Untuk bisa menggunakan Mongodb versi 3 di Raspberry Pi, dibutuhkan *compile* kode sumber dari Mongodb. Pada penelitian ini akan digunakan *file* Mongodb yang telah disesuaikan untuk arsitektur ARM yang digunakan Raspberry Pi, oleh Andy Felong, yang tersedia pada *website* http://andyfelong.com. Selanjutnya dibuat struktur dokumen yang akan digunakan seperti yang ditunjukkan pada Gambar 5.19.



Gambar 5.19 Struktur dokumen yang digunakan dalam *server*

Kode perintah pembuatan *database*

```
1  mongo
2  use pan6_sensor
3  db.createUser({
4                 user:"pi",
5                 pwd:"pan6",
6                 roles:["readWrite", "dbAdmin"]
7  });
8  db.createCollection('sensorData');
9  exit
```

Tabel di atas adalah kode perintah untuk pembuatan *database.* Baris ke 3 sampai 6 adalah untuk pembuatan *user* yang bisa mengakses *database.* Bari ke 8 adalah pembuatan tabel, atau dalam Mongodb disebut *collection*.

Kode Sumber 1: Cek ketersediaan *interface*

```
1  class Network:
2      def iface_check(ifaces):
3                 if ifaces in netifaces.interfaces():
4                       return True
5                 else:
6                       return False
```

Kode sumber 1 adalah mengenai pengecekan apakah *interface* yang digunakan ada atau bisa diakses. *Interface* yang dimaksud dimasukkan dalam variabel *ifaces*, pada baris 3, dan pengembalian dari fungsi ini adalah *True* jika *interface* tersedia, dan *False* jika sebaliknya. Kode sumber 2 mengenai prosedur *restart interface*. Funsi ini akan dipanggil jika terjadi *error* dalam komunikasi *socket.* Baris ke 3 adalah untuk memberi waktu pada *interface* hingga selesai melakukan inisialisasinya, seperti mendapat alamat *IP*.

Kode Sumber 2: *Restart interface*

```
1         def re_start_iface():
2                 call(["sudo service pan6 restart"])
3                 time.sleep(5)
```

Kode Sumber 3: Ambil alamat *interface*

```
1   def iface_loc(link_avail, link):
2       if link_avail > 1:
3           interface_switch = {"global": 0, "local": 1}
4       else:
5           interface_switch = {"local": 0}
6       for i in range(1):
7           if link in interface_switch:
8               ip_link = interface_switch[link]
9               return ip_link
10  def parse_ip(ifaces, iface_link, addrs, port):
11      ip_get = addrs[netifaces.AF_INET6]
                  [iface_link]["addr"]
12      ip_list = [(ip_get,) + (port,)]
        for addr in ip_list:
            for arg in socket.getaddrinfo(addr[0], addr[1],
13                      socket.AF_INET6, socket.SOCK_STREAM,
                        socket.SOL_TCP):
                ddr_fam, socktype, proto, canonname,
                  sockaddr = arg
14      ip = ip_list[0][0]
15      e = len(ifaces)
16      if ip.endswith('%' + ifaces):
17          e += 1
18          ip = ip[:-e]
19      ip = (ip, int(sockaddr[1]),
              int(sockaddr[2]), int(sockaddr[3]))
20      return ip
21  def get_ip(ifaces, link, port):
22      while True:
23          addrs = netifaces.ifaddresses(ifaces)
24          link_avail = len(addrs[netifaces.AF_INET6])
25          while True:
26              iface_link = Network.iface_loc
                             (link_avail, link)
27              if iface_link is None:
28                  break
29              else:
30                  ip = Network.parse_ip(ifaces,
                         iface_link, addrs, port)
31                  return ip
```

Kode sumber 3 adalah mengenai pengambilan alamat *interface* dengan format *tuple* (*host*, *port*, *flow info, scope id*). Fungsi ini dimulai dari *method* get_ip pada baris 21 sampai 31. Baris ke 23 dan 24 dilakukan pengambilan daftar alamat yang tersedia pada *interface* dengan hasil keluaran seperti pada Gambar 5.20. Selanjutnya dilakukan penentuan alamat yang akan digunakan, apakah menggunakan alamat *global* atau *local*, dalam penelitian ini alamat yang digunakan adalah *global*. Penentuan alamat ini dilakukan dalam *method* iface_loc pada baris 1 sampai 9. Karena hasil keluaran dari variabel link_avail sebelumnya adalah dalam bentuk *list*, dalam *method* iface_loc dilakukan identifikasi lokasi alamat dalam *list* tersebut. Dalam hal ini, sesuai dengan contoh pada Gambar 5.20, urutan lokasi alamat adalah [0, 1, 2, 3]. Sesuai dengan urutan penulisan alamat *interface* dimana alamat *global* akan ditulis terlebih dahulu dan diikuti dengan alamat *local. Method* parse_ip digunakan untuk menguraikan hasil keluaran dalam *list* sebelumnya menjadi *tuple* yang berisi alamat sesuai *link* yang digunakan. Contoh keluaran dari fungsi ini adalah ('fdca:9226:7b1c:df45:24:ff:fe00:2', 5005, 0, 0).

```
link_avail: [{'addr': 'fdca:9226:7b1c:df45:24:ff:fe00:2',
              'netmask': 'ffff:ffff:ffff:ffff::/64'},
             {'addr': 'fdca:9226:7b1c:df45:e01e:2c1f:6166:ad9a',
              'netmask': 'ffff:ffff:ffff:ffff::/64'},
             {'addr': 'fe80::24:ff:fe00:2%lowpan0',
              'netmask': 'ffff:ffff:ffff:ffff::/64'},
             {'addr': 'fe80::e01e:2c1f:6166:ad9a%lowpan0',
              'netmask': 'ffff:ffff:ffff:ffff::/64'}]
```

Gambar 5.20 Contoh daftar alamat *IP* pada *interface*

Kode sumber 4 adalah mengenai fungsi penyimpanan data pada *database.* Karena perangkat lunak *server* ini dikembangkan dengan menggunakan *threading*. Maka untuk pengaksesan data dibutuhkan modul Lock. Ketika ada *thread* yang mengakses data, atau dalam hal ini menjalankan *class* Database, *thread* tersebut akan mengaktifkan RLock() (baris 7) sehingga *thread* lainnya akan menunggu RLock() dilepas (baris 15) untuk bisa menjalankan *class* Database. Hal ini dilakukan untuk mencegah data yang mungkin akan hilang atau rusak karena diakses oleh beberapa *instance* dalam waktu yang sama. *Method* insert pada baris 9 sampai 13 adalah untuk menjalankan *query insert* pada *database.*

Kode Sumber 4: Fungsi *database*

```
1  class Database:
2          def __init__(self, node_id, sensor_data):
3                  self.node_id = node_id
4                  self.sensor_data = sensor_data
5                  self.lock = threading.RLock()
6          def insert(self):
7                  self.lock.acquire()
8                  try:
```

```
9                       client = MongoClient()
10                      db = client["pan6_sensor"]
11                      collections = db.sensorData
12                      insert = {"nodeId": self.node_id, "value":
                                self.sensor_data}
13                      collections.insert_one(insert)
14              else:
15                      self.lock.release()
```

Ketika ada data dari *node* yang masuk, data tersebut akan diproses menggunakan *thread* lainnya, misalnya *Thread-1*, seperti dalam kode sumber 5. *Class* NodeHandler akan dijalankan oleh *thread* lain. Dan fungsi yang dijalankan memanggil (menjalankan) *class* Database. *Thread* yang menjalankan *class database* adalah *thread* yang sama dengan *class* NodeHandler.

Kode Sumber 5: Fungsi penanganan *node*

```
1  class NodeHandler(threading.Thread):
2         def __init__(self, node_data, node_ip):
3                 threading.Thread.__init__(self)
4                 self.data = node_data
5                 self.ip = node_ip
6                 self.size = 2048
7                 self.ip_table = []
8         def run(self):
9                 Database(self.ip, self.data).insert()
```

Kode sumber 6 adalah mengenai fungsi *beacon* yang telah dirancang pada bab sebelumnya. *Class* BeaconHandler ini akan berjalan menggunakan *daemon thread*. *Method* mcast_socket adalah untuk membuat koneksi *socket multicast*. Variabel if_idx digunakan untuk merubah nama *interface* menjadi nomor *index interface* yang digunakan. *Method* get_server_ip digunakan untuk mengambil alamat *IP* (baris 17) lalu merubahnya menjadi format data JSON (baris 18) yang akan dikirim ke *node* (baris 26). Baris ke 21 adalah alamat *multicast* yang digunakan. Baris ke 27 memungkinkan untuk pesan *multicast* dikirim sekali setiap 15 detik, sesuai dengan rancangan pada bab sebelumnya.

Kode Sumber 6: Fungsi *beacon server*

```
1  class BeaconHandler(threading.Thread):
2         def mcast_socket(self):
3                 try:
4                       if_idx = socket.if_nametoindex(ifaces)
5                       mcast_sock = socket.socket(socket.AF_INET6,
                                                   socket.SOCK_DGRAM)
6                       mcast_sock.setsockopt(socket.IPPROTO_IPV6,
                                              socket.IPV6_MULTICAST_IF,
```

```
                                        if_idx)
7                   mcast_sock.setsockopt(socket.SOL_SOCKET,
                                        socket.SO_REUSEADDR, 1)
8                   return mcast_sock
9             except socket.error as err:
10                  if mcast_sock in err:
11                        mcast_sock.close()
12                        raise socket.error
13            except OSError as err:
14                  Network.re_start_iface()
15      def get_server_ip(self):
16            link = "global"
17            server_addr = Network.get_ip(ifaces, link, 5005)
18            server_addr = json.dumps(server_addr)
19            return server_addr
20      def run(self):
21            mcast_addr = 'ff02::1'
22            mcast_port = 42424
23            server_addr = self.get_server_ip()
24            mcast_sock = self.mcast_socket()
25            while True:
26                  mcast_sock.sendto(bytes(server_addr, "utf-8"),
                                        (self.mcast_addr))
27                  time.sleep(15)
```

Class Server pada kode sumber 7 adalah titik awal ketika perangkat lunak *server* dijalankan. Baris 6 dan 7 adalah untuk menjalankan memulai *socket*. Baris 8 sampai 10 adalah menjalankan *class* BeaconHandler() menggunakan *daemon thread*. Pada baris 13 *server* siap untuk menerima data yang dikirim oleh *node* yang selanjutnya akan dialihkan ke *thread* lainnya untuk menjalankan *class* NodeHandler(). *Thread* tersebut akan dimasukkan dalam *list* self.threads (baris 3), yang selanjutnya bisa dihentikan ketika proses yang dijalankan sudah selesai dengan melakukan join() (baris 19 sampai 21). Pada baris 22 dan 23 akan dilakukan *restart interface* ketika terjadi *error* pada komunikasi *socket.*

Kode Sumber 7: Fungsi *server*

```
1   class Server:
2         def __init__(self):
3               self.threads = list()
4         def run(self):
5               link = "global"
6               full_addr = Network.get_ip(ifaces, link, self.port)
7               sock = Network.create_socket(full_addr)
```

```
8               mcast_thread = BeaconHandler()
9               mcast_thread.daemon = True
10              mcast_thread.start()
11              while True:
12                      try:
13                              node_data, addr = sock.recvfrom
                                                  (self.size)
14                              node_data = node_data.decode("utf-8")
15                              node_ip = re.sub(':', '', addr[0])
16                              node_thread = NodeHandler(node_data,
                                                          node_ip)
17                              self.threads.append(node_thread)
18                              node_thread.start()
19                              for node_thread in self.threads:
20                                      node_thread.join()
21                              del self.threads[:]
22                      except OSError as err:
23                              Network.re_start_iface()
24                              continue
```

5.2.5 Implementasi purwarupa perangkat lunak *node*

Fungsi *beacon* pada *node* ini berbeda dengan yang ada di *server*. Kode sumber 8 sampai 11 adalah kode sumber untuk fungsi *beacon* pada *node*. *Method* mcast_socket() pada kode sumber 8 digunakan untuk membentuk komunikasi *socket multicast*. Pada baris ke 3 diambil nomor *index interface* yang digunakan. Selanjutnya pada baris ke 4 untuk konfigurasi *group multicast.* Variabel *group* ini akan menghasilkan data berbentuk *binary* yang berisi *address family* dan *IP* (yang disusun dari alamat *interface multicast* yang digunakan, ditambah dengan nomor *index interface).*

Kode Sumber 8: Fungsi *beacon node*

```
1   class BeaconListener:
2         def mcast_socket(self, ifaces):
3                 if_idx = socket.if_nametoindex(ifaces)
                  group = socket.inet_pton(socket.AF_INET6,
4                                          self.mcast_addr[0]) +
                                           struct.pack("i", if_idx)
5                 try:
6                       mcast_sock = socket.socket(socket.AF_INET6,
                                                   socket.SOCK_DGRAM)
7                       mcast_sock.setsockopt(socket.IPPROTO_IPV6,
                                              socket.IPV6_JOIN_GROUP,
```

```
                                        group)
8                   mcast_sock.setsockopt(socket.SOL_SOCKET,
                                        socket.SO_REUSEADDR, 1)
9                   mcast_sock.bind(self.mcast_addr)
10                  return mcast_sock
11              except socket.error as err:
12                  if mcast_sock in err:
13                      mcast_sock.close()
```

Kode sumber 9 adalah mengenai fungsi *listening* pada *beacon node*. Dalam fungsi ini digunakan *modul* select() untuk menerima data dari beacon *server* (baris 5 – 8). Dan pada baris 10 dilakukan pembacaan data JSON yang telah diterima.

Kode Sumber 9: Fungsi *listening beacon node*

```
1  def listen(self):
2      mcast_recv = self.mcast_socket(ifaces)
3      input_recv = [mcast_recv]
4      while True:
5          inputready, outputready, exceptready = select.select
                                            (input_recv, [], [], 5)
6          for s in inputready:
7              if s is mcast_recv:
8                  data, sender = mcast_recv.recvfrom(1500)
9                  if data:
10                     beacon_data = json.loads(data.decode("utf-8"))
11                     return beacon_data
12                 else:
13                     return None
14                 break
```

Kode sumber 10 adalah *starting point* dari fungsi *beacon node*. Dalam fungsi ini dilakukan *listening* untuk "mendengar" *multicast* dari *server* (baris 4). Bagian ini akan terus berjalan hingga ada data yang diterima (baris 5 dan 6). Setelah ada data yang diterima, data tersebut akan disimpan dalam variabel server_addr.

Kode Sumber 10: *Starting point* fungsi *beacon node*

```
1  def run(self):
2      while True:
3         try:
4             beacon_data = self.listen()
5             if beacon_data is None:
6                 continue
7             else:
8                 server_addr = beacon_data
```

```
9                   return server_addr
10              except OSError as err:
11                  Network.re_start_iface()
12                  continue
```

Kode sumber 12 adalah mengenai fungsi pengambilan data. Ada dua fungsi yang tedapat dalam *class* Payload, *method* temperature_data yang digunakan untuk mengambil data dari *sensor* DHT22 dan *method* random_data yang digunakan untuk menghasilkan data secara acak dalam rentang 1 hingga 100 (baris 10). Adanya dua fungsi ini dilakukan karena penggunaan *sensor* DHT22 hanya pada satu *node*, dan *node* lainnya akan menggunakan fungsi random_data. Baris ke 5 digunakan untuk merubah data dari *sensor* yang awalnya adalah banyak angka dibelakang koma, menjadi hanya dua angka dibelakang koma dengan tipe data *float*.

Kode Sumber 12: Fungsi pengambilan data *sensor*

```
1   class Payload:
2       def temperature_data(self):
3               humidity, temperature =
                    Adafruit_DHT.read_retry(self.sensor, self.pin)
4               if humidity is not None and temperature is not None:
5                   data_temp = "{:.2f}".format(temperature)
6                   return data_temp
7               else:
8                   return None
9       def random_data(self):
10              rand_data = str(randint(1, 100))
11              return rand_data
12      def run(self):
13              payload = self.payl_type
14              if payload == "temp":
15                  data = self.temperature_data()
16              else:
17                  data = self.random_data()
18              return data
```

Kode sumber 12 adalah fungsi utama dari perangkat lunak *node*, dimana *class* Node ini adalah yang pertama kali dijalankan. Yang pertama dijalankan adalah *listening multicast* untuk mendapatkan alamat *IP* dari *server* (baris 9). Setelah mendapatkan alamat pengiriman, *class* Payload akan dijalankan untuk mendapatkan data dari *sensor* untuk dikirim, dengan jeda antar pengambilan data selama 2 detik (baris 13 sampai 17). *Method* send_data pada baris 2 sampai 6 akan dipanggil pada baris 16. Fungsi pengambilan dan pengiriman ini akan berhenti setiap 15 detik (baris 11 dan 12), untuk melakukan *listening* ulang untuk mengetahui apakah alamat server masih sama.

Kode Sumber 12: Fungsi pengiriman data *node*

```
1  class Node:
2   def send_data(self, node_data, node_sock, send_address):
3    if type(node_data) is bytes:
4      node_sock.sendto(str(node_data, "utf-8"), send_address)
5    else:
6      node_sock.sendto(bytes(node_data, "utf-8"), send_address)

7   def run(self):
8    while True:
9      send_address = BeaconListener().run()
10       if send_address is not None:
11         send_time = time.time() + 15
12         while time.time() < send_time:
13           node_data = Payload(payload_type).run()
14           if node_data is not None:
15             try:
16               self.send_data(node_data, node_sock, send_address)
17               time.sleep(2)
18               continue
19             except OSError as err:
20               Network.re_start_iface()
21               break
22           else:
23             continue
24         continue
25       else:
26         continue
```

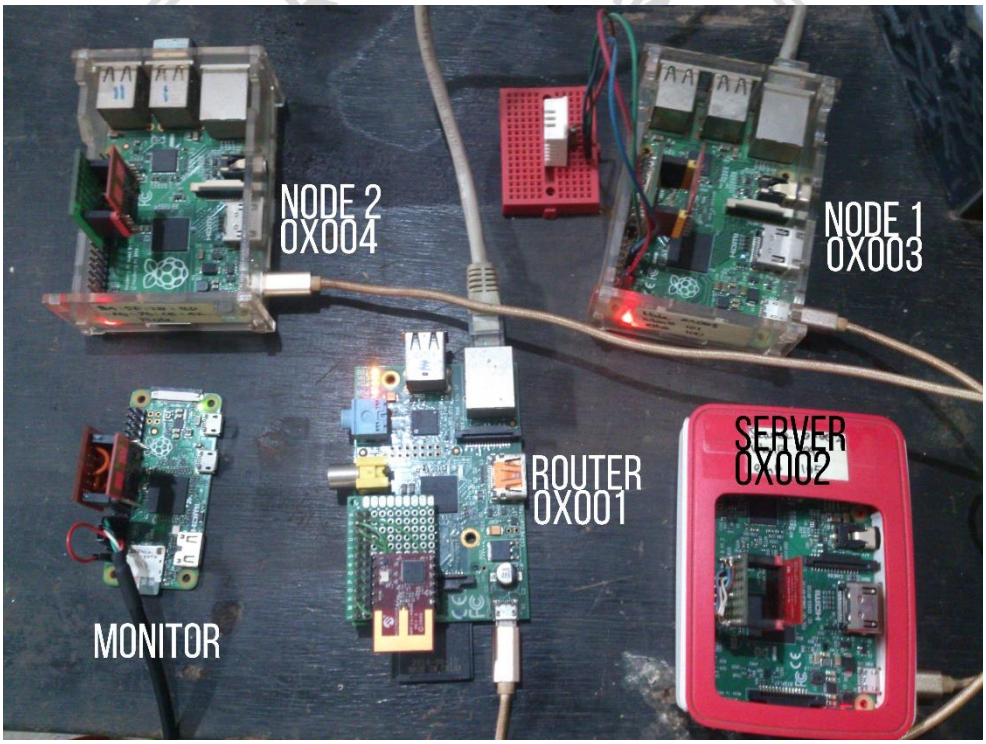
BAB 6 PENGUJIAN DAN ANALISIS

Bab ini membahas mengenai pengujian dari implementasi yang telah dilakukan dan analisis mengenai hasil pengujian. Pengujian yang dilakukan akan terbagi sesuai dengan fitur dan fungsi dari sistem yang telah dibahas pada bab kebutuhan fungsional.

6.1 Pengujian MRF24J40MA sebagai *6LoWPAN radio*

6.1.1 Tujuan

Pengujian perangkat keras MRF24J40MA sebagai *6LoWPAN radio* ini dilakukan untuk mengetahui apakah implementasi pada perangkat keras yang telah dilakukan sebelumnya telah bekerja sesuai dengan keinginan. Dalam hal ini apakah modul *radio* MRF24J40MA bisa digunakan sebagai media komunikasi dalam sistem ini. Gambar 6.1 menunjukkan perangkat keras dari sistem yang digunakan dalam pengujian, yang terdiri dari dua *node*, satu *router*, satu *server,* dan satu *monitor*.



Gambar 6.1 Perangkat keras sistem yang digunakan untuk pengujian

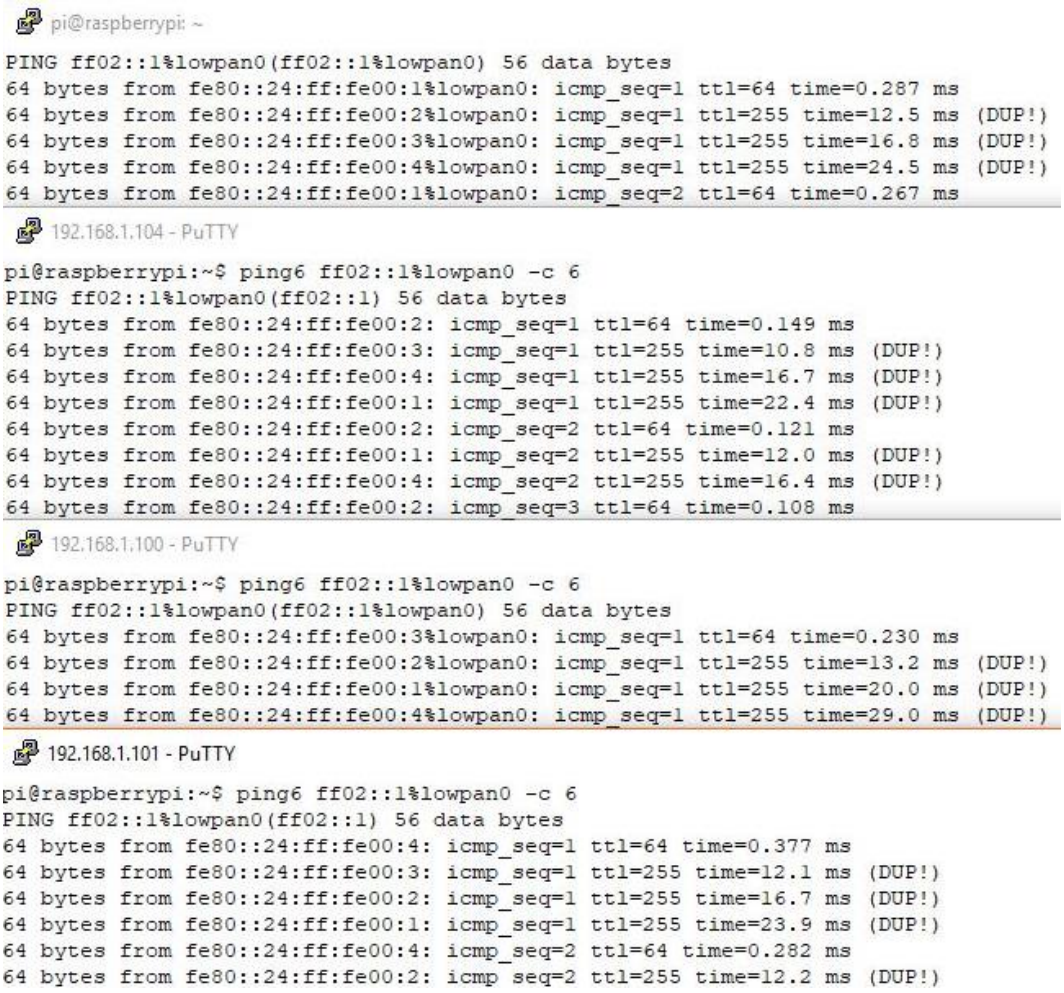
6.1.2 Prosedur pelaksanaan pengujian

Pada pengujian MRF24J40MA ini, parameter pengujian yang diberikan adalah apakah MRF24J40MA bisa digunakan sebagai media komunikasi menggunakan protokol 6LoWPAN. Prosedur pengujian yang dilakukan adalah sebagai berikut:

1. Satu perangkat keras Raspberry Pi dan MRF24J40MA dijalankan dalam *mode monitor* untuk melakukan *sniffing* paket.

2. Masing-masing perangkat keras Raspberry Pi (dengan jumlah 4) melakukan *ping6* ke *multicast* untuk mengetahui apakah semua perangkat keras merespon *ping* tersebut.
3. Hasil *sniffing* dari masing-masing perangkat akan dianalisa menggunakan perangkat lunak Wireshark untuk mengetahui apakah protokol yang digunakan adalah *6LoWPAN.*

6.1.3 Hasil Pengujian



Gambar 6.2 Hasil pengujian dengan *ping6*

Pengujian yang dilakukan dengan melakukan *ping6* ke alamat *multicast* (ff02::1), seperti yang terlihat pada Gambar 6.2, menunjukkan bahwa masing-masing perangkat keras dapat berkomunikasi satu sama lain.

6.2 Pengujian *6LoWPAN router*

6.2.1 Tujuan

Pengujian fungsi perangkat keras sebagai *6LoWPAN router* ini dilakukan untuk mengetahui apakah *router* sudah berjalan dengan semestinya dan mampu memberikan alamat *IPv6* kepada *node* dengan cara *stateless autoconfiguration*.

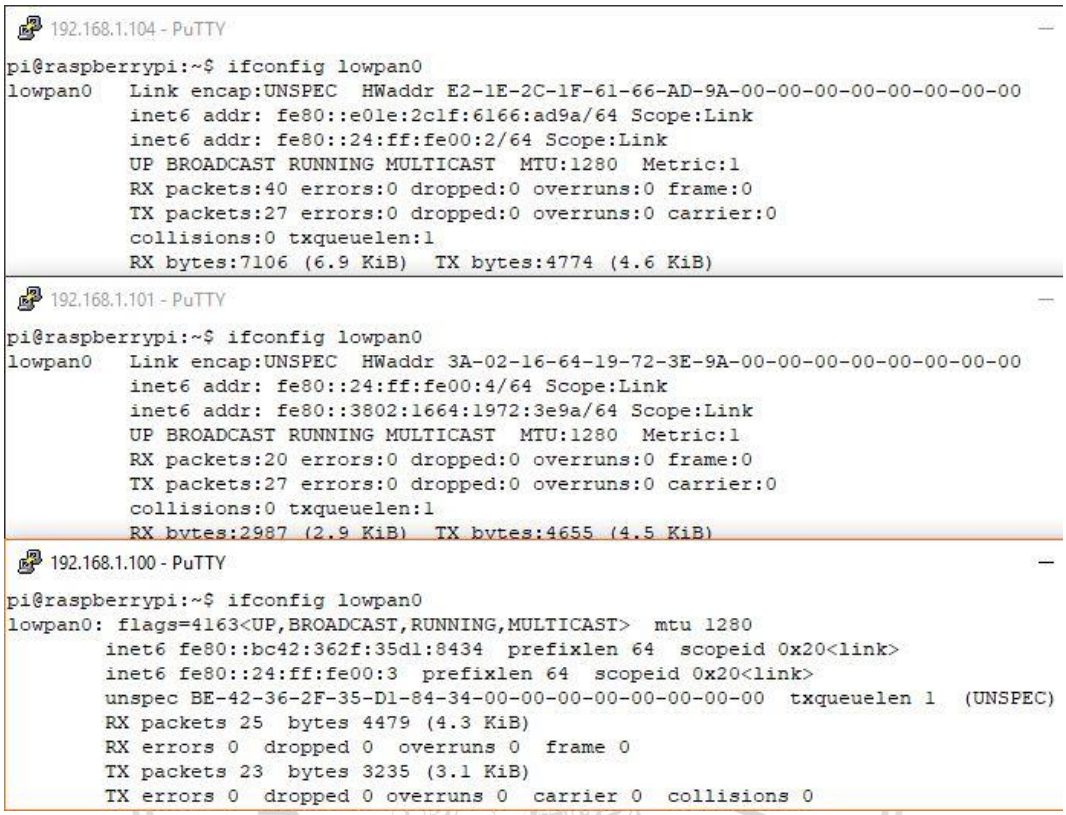
Dengan cara *stateless autoconfiguration* yang dimaksud adalah apakah *router* bisa menjawab pesan *router solicitation* yang dikirim oleh *node* dengan pesan *router advertisement*.

6.2.2 Prosedur pelaksanaan pengujian

Pada pengujian fungsi *router* ini, parameter yang diberikan adalah apakah *node* bisa mendapatkan alamat *IPv6 global* dari *router.* Prosedur pengujian yang dilakukan adalah sebagai berikut:

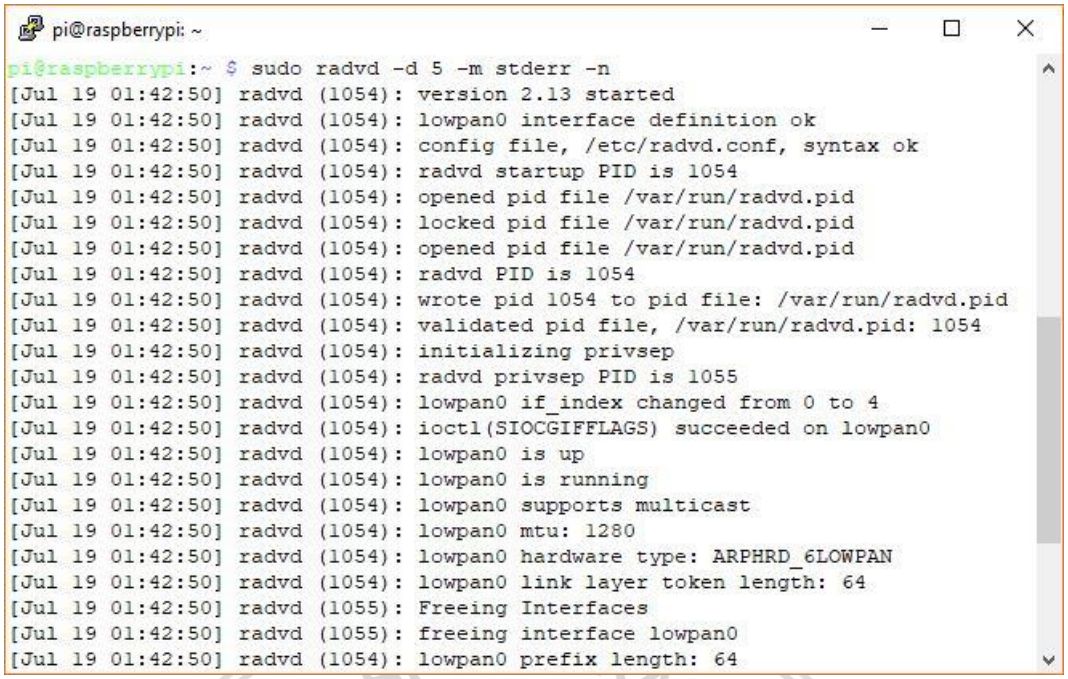
1. Salah satu perangkat keras Raspberry Pi menjalankan *radvd* dalam *debug mode.*
2. Satu perangkat keras Raspberry Pi dan MRF24J40MA dijalankan dalam *mode monitor* untuk melakukan *sniffing* paket.
3. Tiga perangkat keras lainnya akan melakukan *restart interface*, yang dilakukan secara bergantian.
4. Dilakukan perbandingan alamat *interface* lowpan0 pada kondisi awal (belum mendapat *IP*) dengan kondisi setelah dilakukan *restart interfaces,* apakah *interface* sudah mempunyai alam*at global.*
5. Hasil *sniffing* dari masing-masing perangkat akan dianalisa menggunakan perangkat lunak Wireshark untuk mengetahui apakah terjadi pertukaran pesan *router solicitation* dan *router advertisement* antara *router* dengan *host.*

6.2.3 Hasil Pengujian

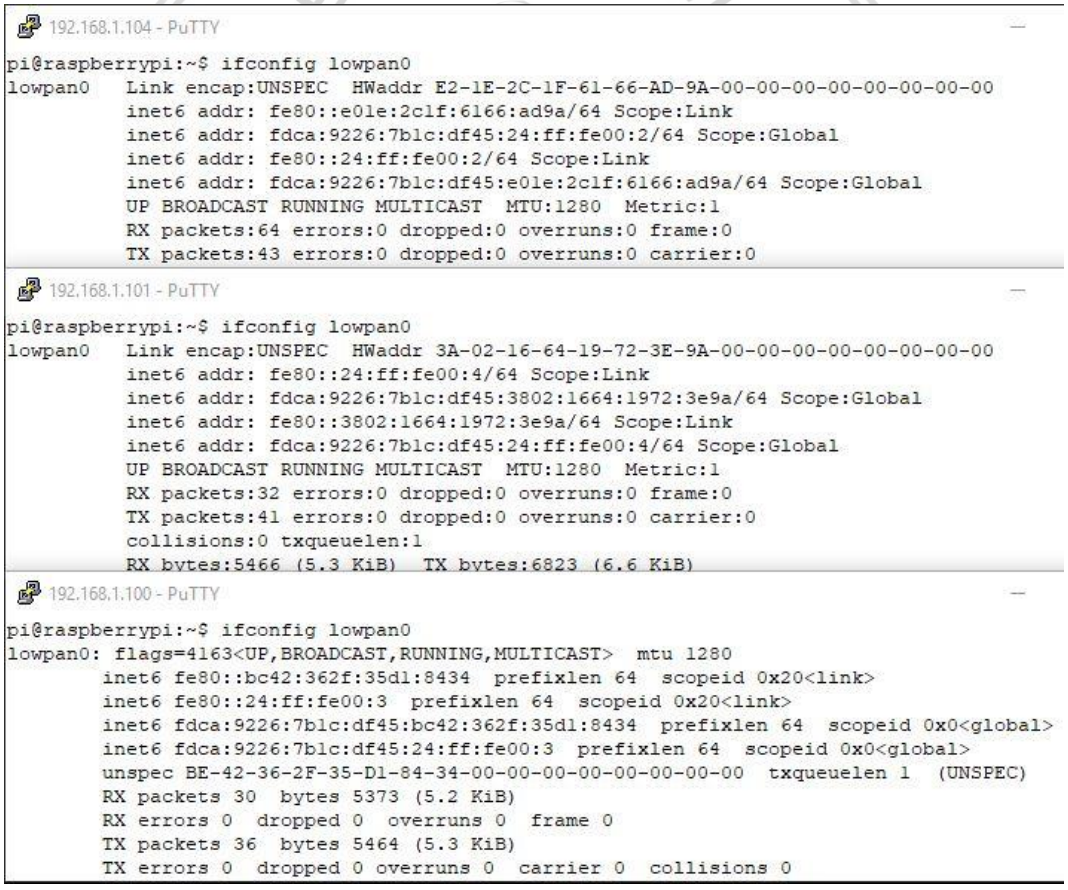


Gambar 6.3 *interface* lowpan0 sebelum mendapatkan *IP global*

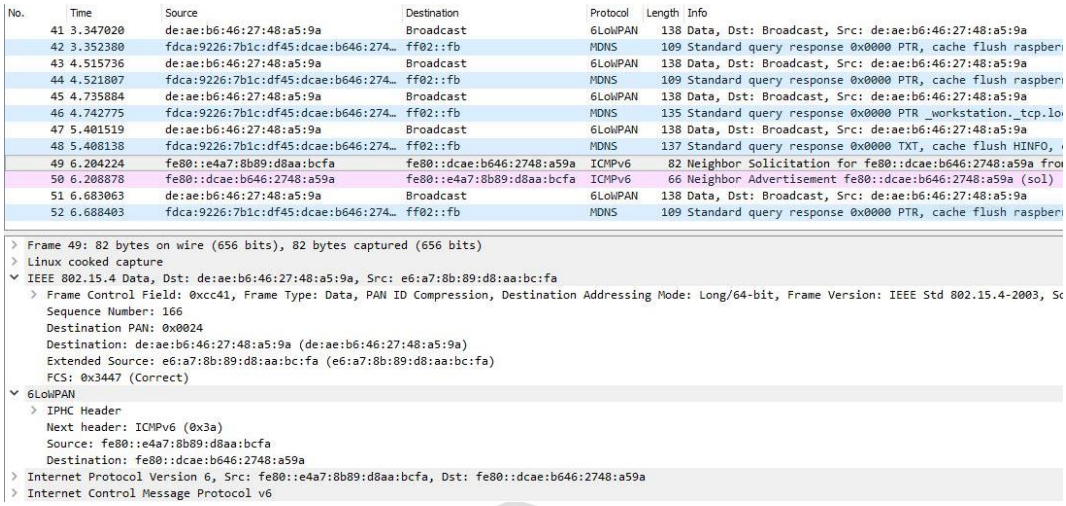
Gambar 6.3 menunjukkan tiga perangkat keras, dengan tampilan *ifconfig* interface lowpan0, dan bisa dilihat bahwa ketiga perangkat tersebut hanya memiliki alamat *IP* link *local.* Setelah *router* dinyalakan, seperti pada Gambar 6.4, ketiga perangkat tersebut mempunyai *IP global*, yang ditunjukkan pada Gambar 6.5. Alamat *IP* yang lebih dari satu, baik itu *local* dan *global*, dikarenakan penulis memberikan dan menggunakan alamat *IP short address* pada semua perangkat, seperti contohnya 0x001. Hasil *sniffing* pada Gambar 6.6, juga menunjukkan adanya pertukaran pesan *router solicitation* dan *router advertisement* antara *router* dengan *host.*



Gambar 6.4 Radvd berjalan dalam mode *debug*



Gambar 6.5 *interface* lowpan0 setelah mendapatkan *IP global*



Gambar 6.6 Hasil *sniffing* menunjukkan pesan *RA* dan *RS*

6.3 Pengujian fitur beacon

6.3.1 Tujuan

Pengujian ini dilakukan untuk mengetahui apakah fitur beacon dapat berjalan sesuai dengan rancangan yang telah dibuat. Fungsi dari fitur beacon adalah untuk memberikan alamat *IP* yang digunakan oleh *server* ke *node* dalam satu *group multicast*. Parameter yang diberikan dalam pengujian ini adalah sebagai berikut:

1. Apakah *server* dapat mengirim alamat *IP* yang digunakan menggunakan *multicast*.
2. Apakah *node* dapat menerima *multicast* dari *server*, dan menggunakan alamat *IP* tersebut sebagai tujuan pengiriman data.

6.3.2 Prosedur pelaksanaan pengujian

Prosedur pengujian yang dilakukan adalah sebagai berikut:

1. Perangkat keras dinyalakan sesuai dengan fungsinya masing-masing (1 *router,* 1 *server*, dan 2 *node*).
2. Perangkat lunak pada perangkat keras dijalankan sesuai dengan fungsinya masing-masing.
3. Perangkat lunak *server* dijalankan selama 30 detik sebelum diberhentikan dan dilakukan perubahan alamat *IP*, hal ini akan diulangi sebanyak 4 kali.
4. Dilakukan perbandingan antara alamat *IP server*, alamat *IP* yang dikirim oleh beacon *server* dan alamat *IP* yang diterima oleh *beacon node*.

6.3.3 Hasil Pengujian

Server - server_log				
2018-08-03 01:48:25,047	server_log	DEBUG	MainThread	Running on interfaces lowpan0
2018-08-03 01:48:25,048	server_log	DEBUG	MainThread	with address fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-03 01:48:25,050	server_log	DEBUG	Thread-1	Beacon started
2018-08-03 01:49:04,247	server_log	DEBUG	MainThread	Running on interfaces lowpan0
2018-08-03 01:49:04,248	server_log	DEBUG	MainThread	with address fdca:9226:7b1c:df45:c030:955d:d2b7:aae0
2018-08-03 01:49:04,269	server_log	DEBUG	Thread-1	Beacon started
2018-08-03 01:49:47,274	server_log	DEBUG	MainThread	Running on interfaces lowpan0
2018-08-03 01:49:47,275	server_log	DEBUG	MainThread	with address fdca:9226:7b1c:df45:e4a7:8b89:d8aa:bcfa
2018-08-03 01:49:47,277	server_log	DEBUG	Thread-1	Beacon started
2018-08-03 01:50:26,883	server_log	DEBUG	MainThread	Running on interfaces lowpan0
2018-08-03 01:50:26,884	server_log	DEBUG	MainThread	with address fdca:9226:7b1c:df45:d8aa:7f5f:378c:2b81
2018-08-03 01:50:26,905	server_log	DEBUG	Thread-1	Beacon started
2018-08-03 01:51:05,689	server_log	DEBUG	MainThread	Running on interfaces lowpan0
2018-08-03 01:51:05,690	server_log	DEBUG	MainThread	with address fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-03 01:51:05,692	server_log	DEBUG	Thread-1	Beacon started
Server - beacon_log				
2018-08-03 01:48:25,051	beacon_log	DEBUG	Thread-1	Beacon send: ["fdca:9226:7b1c:df45:c859:665d:3b83:ba7f", 5005, 0, 0]
2018-08-03 01:49:31,310	beacon_log	DEBUG	Thread-1	Beacon send: ["fdca:9226:7b1c:df45:c030:955d:d2b7:aae0", 5005, 0, 0]
2018-08-03 01:49:47,278	beacon_log	DEBUG	Thread-1	Beacon send: ["fdca:9226:7b1c:df45:e4a7:8b89:d8aa:bcfa", 5005, 0, 0]
2018-08-03 01:50:26,906	beacon_log	DEBUG	Thread-1	Beacon send: ["fdca:9226:7b1c:df45:d8aa:7f5f:378c:2b81", 5005, 0, 0]
2018-08-03 01:51:05,693	beacon_log	DEBUG	Thread-1	Beacon send: ["fdca:9226:7b1c:df45:c859:665d:3b83:ba7f", 5005, 0, 0]

Gambar 6.7 Perbandingan antara alamat *IP server* dan *IP* yang dikirim oleh beacon

Node 1 – node_log				
2018-08-03 01:48:18,569	node_log	INFO	MainThread	Node started
2018-08-03 01:48:18,605	node_log	DEBUG	MainThread	node addr: ('fdca:9226:7b1c:df45:885f:28bd:a676:cea2', 5005, 0, 0)
2018-08-03 01:48:23,612	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-03 01:48:25,036	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-03 01:48:25,037	node_log	DEBUG	MainThread	Time used by beacon: 6.430856227874756
2018-08-03 01:48:34,047	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-03 01:49:00,734	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-03 01:49:04,255	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c030:955d:d2b7:aae0
2018-08-03 01:49:25,284	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c030:955d:d2b7:aae0
2018-08-03 01:49:46,910	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-03 01:49:47,262	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:e4a7:8b89:d8aa:bcfa
2018-08-03 01:50:05,286	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:e4a7:8b89:d8aa:bcfa
2018-08-03 01:50:26,887	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:d8aa:7f5f:378c:2b81
2018-08-03 01:50:44,912	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:d8aa:7f5f:378c:2b81
2018-08-03 01:51:05,674	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-03 01:51:23,701	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
Node 2 – node_log				
2018-08-02 18:48:22,675	node_log	INFO	MainThread	Node started
2018-08-02 18:48:22,772	node_log	DEBUG	MainThread	node addr: ('fdca:9226:7b1c:df45:6099:e1b0:36de:c177', 5005, 0, 0)
2018-08-02 18:48:25,069	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-02 18:48:43,095	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-02 18:49:03,167	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-02 18:49:04,289	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c030:955d:d2b7:aae0
2018-08-02 18:49:22,314	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c030:955d:d2b7:aae0
2018-08-02 18:49:42,377	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-02 18:49:47,296	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:e4a7:8b89:d8aa:bcfa
2018-08-02 18:50:05,320	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:e4a7:8b89:d8aa:bcfa
2018-08-02 18:50:25,388	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-02 18:50:26,921	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:d8aa:7f5f:378c:2b81
2018-08-02 18:50:44,947	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:d8aa:7f5f:378c:2b81
2018-08-02 18:51:05,011	node_log	DEBUG	MainThread	no beacon received, retrying ... 1
2018-08-02 18:51:05,709	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f
2018-08-02 18:51:23,736	node_log	DEBUG	MainThread	server address: fdca:9226:7b1c:df45:c859:665d:3b83:ba7f

Gambar 6.8 Alamat *IP* yang diterima oleh beacon *node*

Gambar 6.7 menunjukkan alamat *IPv6* yang digunakan oleh *server* dan alamat *IP* yang dikirimkan secara *multicast* oleh *beacon server*. Sedangkan pada Gambar 6.8 menunjukkan alamat *IP* yang diterima oleh *beacon node*. Dengan membandingkan hasil dari 2 gambar tersebut, bisa ditarik kesimpulan bahwa fitur beacon telah berjalan dengan sebagaimana mestinya. Dua *node* yang digunakan mempu menerima alamat *IP* yang dikirim oleh beacon *server*.

6.4 Pengujian pengiriman data *node* ke *server*

6.4.1 Tujuan

Pengujian ini dilakukan untuk mengetahui apakah fitur pengiriman data yang telah diimplementasikan telah sesuai dengan rancangan yang dibuat. Pengujian ini juga akan melihat apakah fitur penyimpanan data menggunakan Mongodb bisa berjalan dengan baik.

6.4.2 Prosedur pelaksanaan pengujian

Dalam pengujian ini, parameter yang diberikan adalah apakah data yang dikirim oleh *node* bisa diterima dan disimpan dalam *database* oleh server. Prosedur pengujian yang dilakukan adalah sebagai berikut:

1. Perangkat keras dinyalakan sesuai dengan fungsinya masing-masing (1 *router,* 1 *server*, dan 2 *node*).
2. Perangkat lunak pada perangkat keras dijalankan sesuai dengan fungsinya masing-masing.
3. Dilakukan perbandingan antara data yang didapatkan oleh *node*, data yang dikirimkan oleh *node*, data yang diterima oleh *server*, dan data yang tersimpan dalam *database*.

6.4.3 Hasil Pengujian

	MongoDB - sensorData		
Time	_id	nodeId	value
2018-1-14 23:27:54	ObjectId(5a5b850a74fece12f1092b57)	fdca92267b1cdf45c030955dd2b7aae0	26.5
2018-1-14 23:27:56	ObjectId(5a5b850c74fece12f1092b59)	fdca92267b1cdf45c030955dd2b7aae0	26.4
2018-1-14 23:27:59	ObjectId(5a5b850f74fece12f1092b5b)	fdca92267b1cdf45c030955dd2b7aae0	26.4
2018-1-14 23:28:01	ObjectId(5a5b851174fece12f1092b5d)	fdca92267b1cdf45c030955dd2b7aae0	26.4
2018-1-14 23:28:06	ObjectId(5a5b851674fece12f1092b5f)	fdca92267b1cdf45c030955dd2b7aae0	26.5

Node - payload_log				
Time	Name	Level	ThreadName	Activity
14 23:27:53,996	payload_log	DEBUG	MainThread	Temp=26.5* Humidity=93.9%
14 23:27:53,998	payload_log	DEBUG	MainThread	Time used by sensor: 0.543989896774292
14 23:27:56,536	payload_log	DEBUG	MainThread	Temp=26.4* Humidity=94.2%
14 23:27:56,537	payload_log	DEBUG	MainThread	Time used by sensor: 0.5283851623535156
14 23:27:59,070	payload_log	DEBUG	MainThread	Temp=26.4* Humidity=94.2%
14 23:27:59,072	payload_log	DEBUG	MainThread	Time used by sensor: 0.5284640789031982
14 23:28:01,608	payload_log	DEBUG	MainThread	Temp=26.4* Humidity=94.2%
14 23:28:01,610	payload_log	DEBUG	MainThread	Time used by sensor: 0.5282597541809082
14 23:28:06,675	payload_log	DEBUG	MainThread	Temp=26.5* Humidity=94.2%
14 23:28:06,678	payload_log	DEBUG	MainThread	Time used by sensor: 3.0615851879119873

Node - data_sent_log				
Time	Name	Level	ThreadName	
14 23:27:54,001	data_sent_log	DEBUG	MainThread	Message sent 26.50
14 23:27:56,539	data_sent_log	DEBUG	MainThread	Message sent 26.40
14 23:27:59,074	data_sent_log	DEBUG	MainThread	Message sent 26.40
14 23:28:01,612	data_sent_log	DEBUG	MainThread	Message sent 26.40
14 23:28:06,681	data_sent_log	DEBUG	MainThread	Message sent 26.50

Server - data_recv_log				
Time	Name	Level	ThreadName	Activity
2018-1-14 23:27:54,027	data_recv_log	DEBUG	Thread-2	Received data from fdca92267b1cdf45c030955dd2b7aae0 : 26.50
2018-1-14 23:27:56,565	data_recv_log	DEBUG	Thread-3	Received data from fdca92267b1cdf45c030955dd2b7aae0 : 26.40
2018-1-14 23:27:59,106	data_recv_log	DEBUG	Thread-4	Received data from fdca92267b1cdf45c030955dd2b7aae0 : 26.40
2018-1-14 23:28:01,637	data_recv_log	DEBUG	Thread-5	Received data from fdca92267b1cdf45c030955dd2b7aae0 : 26.40
2018-1-14 23:28:06,706	data_recv_log	DEBUG	Thread-6	Received data from fdca92267b1cdf45c030955dd2b7aae0 : 26.50

Gambar 6.9 Hasil pengujian pengiriman, penerimaan, dan penyimpanan data

Gambar 6.9 menunjukkan hasil dari pengujian yang dilakukan, dan gambar tersebut hanyalah sebagian dari data yang didapat dari pengujian. Dengan dilakukan perbandingan pada gambar tersebut, bisa dilihat bahwa semua data yang dikirim, diterima, dan disimpan adalah benar. Untuk fitur *beacon* bisa dilihat pada Gambar 6.10 bahwa fitur ini juga berfungsi dengan baik. Dengan data dari gambar 6.9, 6.10, dan Tabel 6.1, penulis bisa menyimpulkan bahwa sistem ini telah mampu unutk menjalankan fungsi dan fitur yang telah dirancang dan diimplementasikan dalam penelitian ini.

Server - beacon_log				
Time	Name	Level	ThreadName	
2018-01-14 23:27:53,463	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:27:54,465	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:27:55,467	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:27:56,469	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:27:57,471	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:27:58,474	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:27:59,481	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:00,484	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:01,486	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:02,489	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:03,491	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:04,498	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:05,500	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:06,503	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:07,505	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:08,507	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:09,512	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:10,514	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:11,517	beacon_log	DEBUG	Thread-1	Beacon send
2018-01-14 23:28:12,520	beacon_log	DEBUG	Thread-1	Beacon send

Gambar 6.10 Pengujian fitur *beacon*

Tabel 6.1 Hasil pengujian fungsionalitas sistem

No	Fungsi (fitur)	Hasil Uji
1	Beacon	Berfungsi
2	Pengiriman data	Berfungsi
3	Penerimaan data	Berfungsi
4	Penyimpanan data	Berfungsi

BAB 7 PENUTUP

Bab ini akan membahas mengenai kesimpulan yang bisa diperoleh setelah dilaksanakannya penelitian ini, serta saran yang penulis berikan untuk pengembangan dan penelitian selanjutnya.

7.1 Kesimpulan

Berdasarkan dari hasil analisa pengujian yang telah dilakukan sebelumnya, dapat ditarik kesimpulan sebagai berikut:

1. Perancangan pada sistem purwarupa ini digunakan perangkat keras yang terdiri dari Raspberry Pi sebagai *control unit* dan MRF24J40MA sebagai *6LoWPAN radio transceiver*. Perangkat keras yang digunakan ada 4 dengan masing-masing fungsi yang terdiri dari *router, server,* dan 2 *nodes*. Perangkat lunak yang digunakan sebagai *router* adalah RADVD.
2. Agar node bisa mengetahui alamat *IP server* tanpa perlu konfigurasi secara manual, diberikan fitur beacon pada perangkat lunak *server* dan *node*. Cara kerja dari fitur ini adalah, *server* akan mengirim alamat *IP* yang digunakannya secara periodik melalui *multicast,* data yang dikirim melalui *multicast* inilah yang akan diterima dan digunakan sebagai alamat pengiriman data oleh *node*.
3. Hasil yang didapat dari sistem yaitu, *server* mampu menerima data dari *node*, menyimpan data menggunakan NoSQL, fitur beacon pada *server* mampu mengirim alamat *IP* yang digunakan *server* melalui *multicast* dan berjalan menggunakan protokol *6LoWPAN*. Node mampu mengirim data ke server, fitur beacon pada *node* mampu menerima dan menggunakan alamat *IP* yang diterima melalui *multicast* dan berjalan menggunakan protokol *6LoWPAN.* Perangkat keras *router* mampu memberikan alamat *IPv6* pada *host* dengan cara *Stateless Autoconfiguration*.

7.2 Saran

Ada beberapa saran yang bisa penulis berikan untuk pengembangan dari sistem selanjutnya, antara lain:

1. Penggunaan *node* yang lebih banyak untuk menguji kehandalan sistem dalam berbagai situasi.
2. Penambahan fitur *database*, yang tidak hanya untuk menyimpan data.
3. Penambahan antarmuka untuk *end-user*, sehingga pengguna bisa melihat hasil data yang ada.
4. Penggunaan perangkat keras yang berbeda arsitektur untuk mengetahui *interoperability* dari sistem.

DAFTAR PUSTAKA

Ahmed, R. M., Huang, X., Sharma, D. & Cui, H., 2012. Wireless Sensor Network: Characteristics and Architectures. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering,* 6(12).

Babatunde, O. & Al-Debagy, O., 2014. A Comparative Review Of Internet Protocol Version 4 (IPv4) and Internet Protocol Version 6 (IPv6). *International Journal of Computer Trends and Technology (IJCTT),* Volume 13, pp. 10-13.

Bouvy, M., 2013. *Raspberry PI + Xbee: UART / Serial howto.* [Online] Available at: https://michael.bouvy.net/blog/en/2013/04/02/raspberry-pi-xbee-uart-serial-howto/ [Accessed 7 February 2016].

Bröring, A. et al., 2011. New Generation Sensor Web Enablement. *Sensors,* Volume 2652-2699, p. 11.

Connoly, S., 2012. *7 KEY DRIVERS FOR THE BIG DATA MARKET.* [Online] Available at: https://hortonworks.com/blog/7-key-drivers-for-the-big-data-market/ [Accessed 29 December 2017].

Corbet, J., Rubini, A. & & Kroah-Hartman, G., 2005. *Linux Device Drivers.* 3rd ed. Sebastopol, California: O'Reilly Media.

Deru, L. et al., 2013. Redundant Border Routers for Mission-Critical. *Proceedings of the Fifth Workshop on Real-World Wireless Sensor Networks.*

IEEE Computer Society, 2011. Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Standard for Local and metropolitan area networks,* Issue 2011.

Internet Engineering Task Force (IETF), 2017. *UDP Usage Guidelines.*

Kumar, K., 2013. Study of Interrupts (Microprocessor). *International Journal of Technical Research (IJTR),* 2(3), pp. 18-20.

Kushalnagar, N., Montenegro, G. & Schumacher, C., 2007. *http://www.hjp.at/doc/rfc/rfc4919IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals.* [Online] Available at: http://www.hjp.at/doc/rfc/rfc4919.html [Accessed 7 February 2016].

Mayer, K. & Fritsche, W., 2006. IP-enabled Wireless Sensor Networks and Their Integration Into the Internet. *First International Conference on Integrated ad-hoc and Sensor Networks.*

Moniruzzaman, A. B. M. & Hossain, S. A., 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application,* Volume 6.

Mulligan, G., 2007. The 6LoWPAN architecture. *EmNets '07 Proceedings of the 4th workshop on Embedded networked sensors,* pp. 78-82.

Olsson, J., 2014. *6LoWPAN demystified.* [Online] Available at: http://www.ti.com/lit/wp/swry013/swry013.pdf [Accessed 1 February 2016].

Ott, A., 2012. *Wireless Networking with IEEE 802.15.4 and 6LoWPAN.* s.l., Embedded Liinux Conference.

Perera, C., Liu, C. H., Jayawardena, S. & Chen, M., 2014. A Survey on Internet of Things From Industrial Market Perspective. Volume 2, pp. 1660-1679.

Petazzoni, T., 2016. *Device Tree for Dummies.* s.l.:FreeElectrons.

Ramírez, A. N., 2014. *Internet of things implementation with Raspberry Pi*.

Raspberry Pi, 2015. *GPIO: MODELS A+, B+ AND RASPBERRY PI 2.* [Online] Available at: https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/README.md [Accessed 7 February 2016].

Sandeep, V. et al., 2015. Globally accessible machine automation using Raspberry pi based on Internet of Things. *Advances in Computing, Communications and Informatics (ICACCI),* pp. 1144-1147.

Schmidt, S., 2015. *Status Report for IEEE 802.15.4 and 6LoWPAN in Linux.* San Jose.

Shelby, Z., Chakrabarti, S., Nordmark, E. & & Borman, C., 2012. *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs).*

Srinivas, A., Kumar, M. K. & Bhandari, J. K., n.d. Design and Verification of Serial Peripheral Interface. *International Journal of Engineering Development and Research (IJEDR),* pp. 130-136.

Veen, J. S. V. D., Waaij, B. V. D. & Meijer, R. J., 2012. Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. *IEEE Fifth International Conference on Cloud Computing.*

Wood, A., 2015. *The internet of things is revolutionising our lives, but standards are a must.* [Online] Available at: http://www.theguardian.com/media-network/2015/mar/31/the-internet-of-things-is-revolutionising-our-lives-but-standards-are-a-must [Accessed 7 February 2016].

Yick, J., Mukherjee, B. & Ghosal, D., 2008. Wireless sensor network survey. *Computer Networks,* Volume 52.

Zheng, J. & Jamalipour, A., 2009. *Wireless Sensor Network A Networking Perspective.* Hoboken, New Jersey: John Wiley & Sons, Inc.